

ON THE FRONT LINE OF GAME INNOVATION

Game

DEVELOPER

AUGUST 1997

Artists and Programmers in Peaceful Coexistence

HOW TO
FOSTER HARMONY
THROUGH BETTER
COMMUNICATION

Multiplayer Games:

**CONQUERING DATA
DISTRIBUTION
USING GROUPINGS**

PAGE
42

**TRICKS FOR REALISTIC
CHARACTER ANIMATION**

PAGE
50

**DESIGN DOCUMENTS:
A ROADMAP FOR SUCCESS**

PAGE
58

**DANI BUNTEN BERRY
ON THE SOAPBOX**

PAGE
68

*****3-DIGIT 945
951399013 GD 004 120 9610 AUG 97
L C SHADOW DESIGNS
4297 WILSON LA
CA 945521

95 CANADA \$6.95



0 73361 87387 3
DISPLAY UNTIL SEPT. 8, 1997

GO AHEAD.

SMOKE

IN THE

OFFICE.




THE
500
MHZ
ALPHA
CHIP

Fire up a PC powered by an Alpha processor and you'll burn through your workload like never before—because you'll be running your Windows NT® software on the world's fastest microprocessor. Alpha PCs run your favorite Windows® software too. And they're available, at competitive prices, from a variety of manufacturers right now. To see why there's simply no match for Alpha, call (888) ALPHA-45 today. Or see us at www.alphapowered.com.

MICROSOFT
WINDOWS NT
COMPATIBLE

©1997 Digital Equipment Corporation. DIGITAL, the DIGITAL logo and AlphaPowered are trademarks of Digital Equipment Corp. Mitsubishi is a registered trademark of Mitsubishi Electric Corp. Samsung is a trademark of Samsung Electronics Co. Windows and Windows NT are registered trademarks of Microsoft Corp.

digital™

 MITSUBISHI

 SAMSUNG

Alpha processors are made by these leading technology companies.

AlphaPowered™



See Us at Booth #1353,
August 5-7, 1997
Los Angeles Convention Center



"Robot" from "Pegasus Prime" © 1996 Presto Studios, Inc.

ARE YOU GAME?

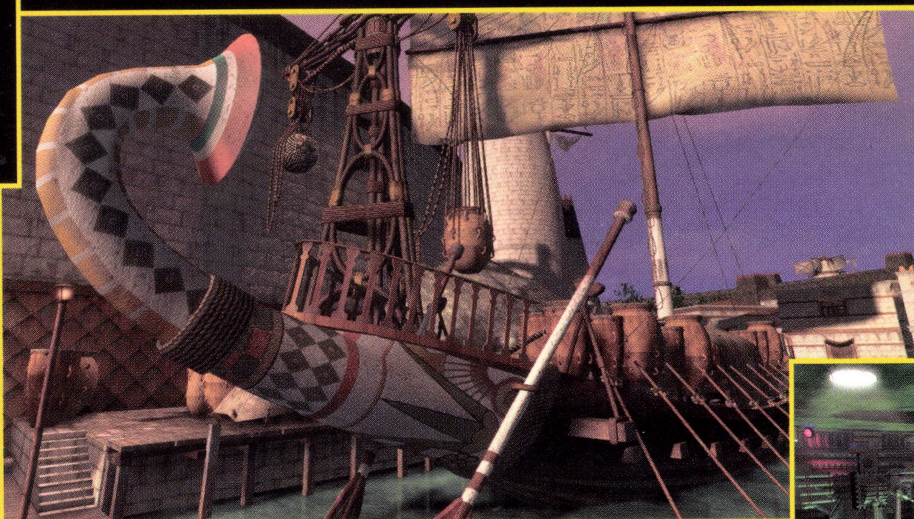
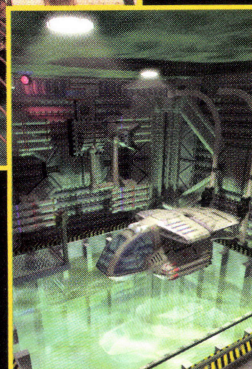


Image from The Journeyman Project 3. © 1997 Presto Studios, Inc.



Image from The Journeyman Project 3.
© 1997 Presto Studios, Inc.



"Norad Sub Room" from
"Pegasus Prime" © 1996
Presto Studios, Inc.

ElectricImageTM
Broadcast
modeling • rendering • animation
\$2,995!

New Decade • New Software • New Platforms

Powerful new features. Dazzling image quality. The world's fastest renderer. Our first ten years were great — our next ten will be awesome.

We're celebrating our diamond anniversary with **new platforms** (like NT and SGI ...), **new products** (like the modeler, radiosity ...) and more. Want the details? Check out our web site www.electricimage.com.

For a **free demo** version of ElectricImage, call now (888) RENDER1.

ELECTRIC IMAGE
INCORPORATED

WORK HARD, RENDER FAST, RETIRE YOUNG.TM

888 RENDER 1 • sales@electricimage.com • www.electricimage.com

Games: The Dark Eye, Bad Mojo, Star Wars: Shadows of the Empire, QIN: Tomb of the Middle Kingdom, Gadget, The Journeyman Project: Buried In Time, The Resident's Bad Day On The Midway, Prince Interactive, Iron Helix, Pegasus Prime.

Broadcast Effects: Dateline NBC, X Files, Fox Television, Sci-Fi Channel, E! Entertainment Television, Nickelodeon, Babylon 5, Sliders, Star Trek: Deep Space Nine, The Real Adventures of Jonny Quest, JAG and many more....



"Moth Flight" From Bad MojoTM and © Pulse Entertainment, Inc.
All rights reserved.



*ElectricImage Broadcast delivers up to NTSC and PAL resolution for television, video and multimedia projects, and high resolution stills (up to 16Kx16K). ElectricImage and Work Hard, Render Fast, Retire Young are trademarks of Electric Image Inc. All other trademarks and copyrights are those of their respective holders. Specifications subject to change without notice.

AUGUST 1997

FEATURES

26 The Artist Synapse

Using a hypothetical Q&A between an RT3D artist and other team members, Josh White translates technical requirements into artist-speak.

BY JOSH WHITE

42 Using Groupings for Networked Gaming

Grouping allows networked games to route essential data among the players while making efficient use of both network and CPU resources. Jesse Aronson guides the reader through efficient data distribution.

BY JESSE ARONSON

50 Using a Base-Root System for Animated Characters

Give your character realistic movement by locking it into a real-world coordinate system. Zed's not dead, baby!

BY SCOTT CORLEY

58 Creating a Great Design Document

A first-class design document captures the body and soul of your game. Here's how to ensure that your vision endures through the development cycle.

BY TZVI FREEMAN

COLUMNS

12 Graphic Content

BY BRIAN HOOK

Multipass Rendering and the Magic of Alpha Blending

21 Artist's View

BY DAVID SIEKS

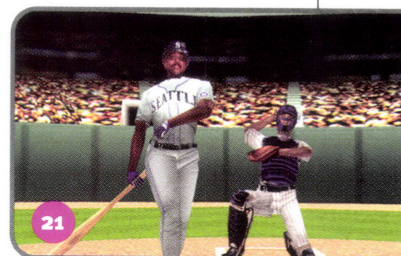
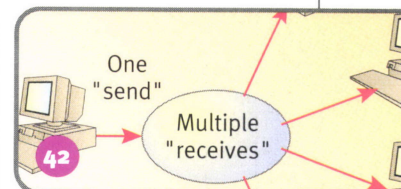
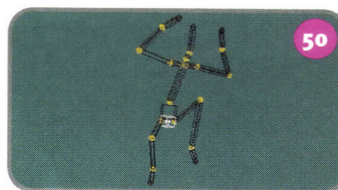
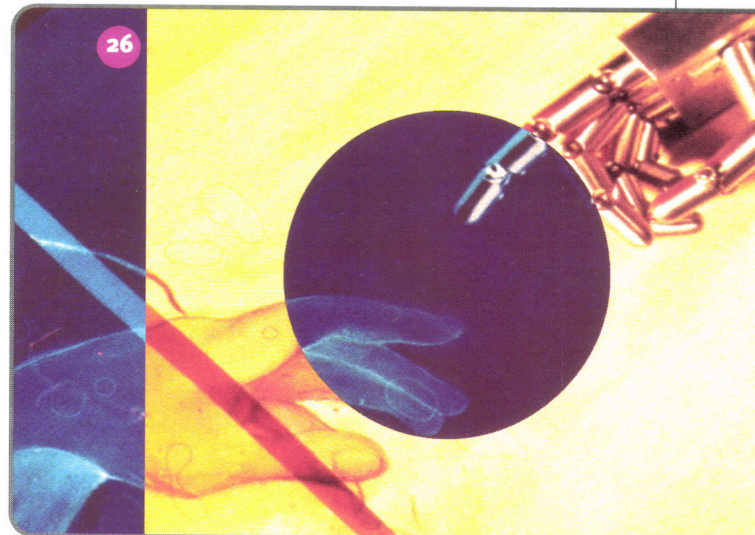
Take Me Into the Ballgame

64 Soapbox

BY DANI B. BERRY

Escaping the Solo Trap

COVER: The cover image was created by Pyros Pictures of Newport Beach, Calif., in 3D Studio MAX 1.2 and composited in PhotoShop 4.0.1. It was originally created as a 30-second animation for their own company logo. See <http://www.grubba.com/pyros> for more of their artwork.



DEPARTMENTS

6 Game Plan

BY ALEX DUNNE

8 Says You!

LETTERS FROM OUR READERS

10 Bit Blasts

Provoking Microsoft, new Ray Dream Studio, wares for tracking your multimedia goods, ActiveX controls, and a 3D game builder from Switzerland.

67 Creative Careers

What are the best jobs in the field?



TURN

Allow us to introduce you to Pyramid3D™, the most powerful
PC graphics processor in the world.

Any world.

You see, Pyramid3D produces the
kind of stunning visual realism
you've only seen on expensive
workstations. That's because
it uses a unique program-
mable, optics-based
architecture that
treats light the same
way nature does,
with all the subtleties the
human eye normally picks up.

Which means you can now bring a whole

YOUR NEXT PC DESIGN INTO A MONSTER HIT.

new world of reality to PC graphics. With advanced
effects like radiosity, Phong shading, bump mapping,
nonlinear fog and perspective-correct texture
and color shading.

All of which can be executed
in real time.

And unlike many chips
that began as 2D accelerators,
Pyramid3D was designed from
the ground up as a 3D engine.

So it's easier to program. And it runs faster than
a raptor on steroids.

Games and professional animations clock
in at a whopping 1 million triangles/sec*. All with a fill rate of
up to 50 million pixels/sec.



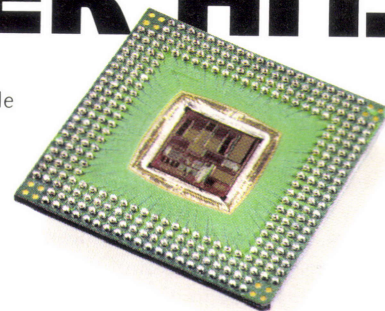
REALITY CHECK	TriTech Pyramid3D	3Dfx Voodoo Rush	REAL3D R3D100	3Dlabs GLINT Gamma + Permedia
Radiosity Texturing**	✓			
Phong Enhanced Texturing**	✓			
Bump Mapping	✓			
Non-linear Fog	✓			
Perspective-Correct Texture, Color, Fog, Transparency	✓			
Direct3D API, OpenGL & Heidi Support	✓	✓	✓	✓
Independent Colored Highlights	✓			
Multiple Independent Textures**	✓			
Object/Vertex Level Processing	✓			✓
Symmetric Multiprocessor Scalability (2 to 10 Processors)	✓			
2D/Video Acceleration	✓			✓
24-bit Z buffer depth	✓		✓	

** Single Pass

It's also compatible
with Direct3D™
and OpenGL™ for
Microsoft® Windows®
platforms.

But this is just a preview. Get the big picture at
www.tritechmicro.com/p3d. Or call 1-888-253-8900,
Ext. 304, to qualify for our design kit.

We'll help you find a whole new
audience for your work.



TriTech
Microelectronics

Debabelizing Development

Seasoned software developers have likely heard of Fred Brooks' book, *The Mythical Man-Month*. In his book, Brooks forwarded the notion (now commonly accepted as gospel) that adding developers to a project already behind schedule can backfire. New people require time and training to get familiarized, which requires time and energy from the team members already on the project. So while the productive team members spend their time briefing the newest people on the project, coaching them, overseeing their work, and so on, less work gets done than it would otherwise. One of the lessons that developers and project managers should glean from this is that good communication comes at a cost.

Certainly, game development is not immune to the scenario that Brooks described, and in fact, game development is more prone than most software development projects to suffer from carelessly adding staff without considering the consequences. Brooks' revelation is all the more interesting when you consider that when *The Mythical Man-Month* was published in 1975, computer games were largely unknown to the general public. Computer entertainment consisted primarily of Wumpus hunting and playing PONG. Game development didn't require the intense collaboration between designers, artists, writers, producers, programmers, music composers, audio effects engineers, and the rest of the specialties that we see in game development today. However, now the industry relies on leading-edge technical knowledge and creative input, so good communication between team members is arguably the most important input in game development today.

The most often cited communication gap in game development is the one separating programmers and artists/animators. Since these two groups make up the lion's share of today's game development teams, it's only natural that misunderstandings will occur more frequently at this junction.

Fortunately, almost all of the programmers and artists/animators that

I've met understand the importance of communicating with one another. This was reconfirmed for me at the CGDC last spring when we held two focus groups, one for artists/animators and one for programmers. In each session, I asked whether there was a problem dealing with the other group. While people in both groups said that interdisciplinary communication was challenging, everyone agreed that getting artists, animators, and programmers to see eye to eye was one of the challenges that made game development rewarding. When everyone made an effort to teach and learn from each other, the quality of the game benefited tremendously. One point that has stuck with me was the interest on the programmers' part to educate artists and animators about programming concepts, since that was often a bottleneck in their communication.

(Incidentally, it's important to note that having good communication doesn't imply that there won't be conflict between team members. In one of my favorite software development books, *Dynamics of Software Development*, Jim McCarthy makes this excellent point: "*The only consensus [between team members] worth having is a creative one achieved in the combat of fully engaged intellects.*" In other words, sometimes developing a great product requires conflict between team members in order to release the maximum amount of creativity.)

Which brings us to Josh White's feature this month. Josh is the rare artist and animator who understands a great deal about the programming side of the equation. That gives him special insight into game development and it's what makes his article significant to artists, animators, and programmers. Whether you're in one group or another, you'll learn something from his peek into the psychology of each group and come away with a better understanding of how both groups translate technology into their own terms.

Alex Dunne

Game DEVELOPER

EDITOR IN CHIEF Alex Dunne
adunne@compuserve.com

MANAGING EDITOR Tor Berg
tdberg@sirius.com

EDITORIAL INTERN Alex Clark
alex@mfi.com

EDITOR-AT-LARGE Chris Hecker
checker@bix.com

CONTRIBUTING EDITORS Larry O'Brien
lobrien@msn.com
Brian Hook
bwh@wksoftware.com
David Sieks
gdmag@mfi.com
Ben Sawyer
bensawyer@worldnet.att.net

ART DIRECTOR Azriel Hayes
ahayes@mfi.com

ADVISORY BOARD Hal Barwood
Noah Falstein
Susan Lee-Merrow
Mark Miller
Josh White

COVER IMAGE Pyros Pictures

PUBLISHER KoAnn Vikören

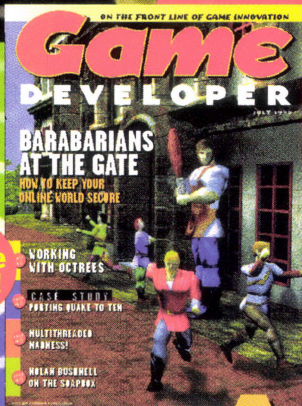
ASSOCIATE PUBLISHER Cynthia A. Blair
(415) 905-2210
cblair@mfi.com

REGIONAL SALES Tony Andrade
MANAGER (415) 905-2156
tandrade@mfi.com

SALES ASSISTANT Chris Cooper
(415) 908-6614
ccooper@mfi.com

MARKETING MANAGER Susan McDonald
MARKETING GRAPHIC DESIGNER Azriel Hayes
AD. PRODUCTION COORDINATOR Denise Temple
DIRECTOR OF PRODUCTION Andrew A. Mickus
VICE PRESIDENT/CIRCULATION Jerry M. Okabe
GROUP CIRCULATION MANAGER Mike Poplarido
SUBS. MARKETING MANAGER Melina Kaplanis
NEWSSTAND MANAGER Eric Alekman
REPRINTS Stella Valdez
(916) 983-6971
un Miller Freeman
A United News & Media publication
CHAIRMAN/CEO Marshall W. Freeman
PRESIDENT/COO Donald A. Pazour
SENIOR VICE PRESIDENT/CFO Warren "Andy" Ambrose
SENIOR VICE PRESIDENTS H. Ted Bahr,
Darrell Denny,
David Nussbaum,
Galen A. Poss,
Wini D. Ragus,
Regina Starr Ridley
VICE PRESIDENT/PRODUCTION Andrew A. Mickus
VICE PRESIDENT/CIRCULATION Jerry M. Okabe
SENIOR VICE PRESIDENT/
SYSTEMS AND SOFTWARE
DIVISION Regina Starr Ridley

www.gdmag.com



ON THE FRONT LINE OF GAME INNOVATION
Game
DEVELOPER

Are You on the Front Line of Game Innovation?

You are if you're reading
Game Developer magazine.

We got a makeover, we've gone
monthly, we're free if you qualify—
and we're still way deep!

Devour These Topics Whole «

3D PROGRAMMING TECHNIQUES
GAME DESIGN
GAME TESTING
ART AND ANIMATION
DEVELOPING NETWORK-BASED GAMES
SOUND DESIGN
DIGITAL ASSET MANAGEMENT

The Full-On Writers «

NOAH FALSTEIN
SID MEIER
CHRIS HECKER
DAVID SIEKS
NOLAN BUSHNELL
CHRIS CRAWFORD
BRIAN HOOK
BEN SAWYER
ANDRE LAMOTHE

Front Page News «

EVERY MONTH OUR COVER WILL FEATURE ART
FROM A TALENTED GAME ARTIST.

Find even more hardcore technical
coverage on programming, 3D and
visual design, sound design, and
game design strategy in *Game
Developer*. It has everything your
entire professional team needs to
bring killer games to market.

Game Developer is FREE to any member of
the professional game development team!
Call 800.250.2429 or stop by www.gdmag.com
and fill out the qualification form for your free
subscription today!

Cool!

Don't Let Go the Code

I'm a "from the first issue" reader of *Game Developer*. I must say this magazine has grown in the right way, and it's very useful in my game programming work. I also liked the new design and your motivations for changing the magazine, but I must admit I was a little disappointed in not seeing a single technical article. Please don't do so!

As an old reader I'm pleased that others involved in game developing could find your magazine new and interesting (artists, designers), but programmers would die without a line of code or some technical hints. You know how weird we game developers are...

STEFANO BARALDI
ITALY

EDITOR ALEX DUNNE RESPONDS:

Don't worry about technical content. In retrospect, it probably wasn't smart of us to schedule an issue on game design that coincided with our relaunch. The lack of code was more coincidental than a planned change in direction away from programming topics. True, we are covering more topics other than programming (such as design, QA, art/animation, and project management), but we don't want to alienate our core readership either.

Our June 1997 issue featured an article about pathfinding AI, this issue has article on networking and programming for 3D animation, inimitable 3D programmer Brian Hook has joined us as a columnist, and upcoming issues have other programming articles as well, with code.

Wherefore the Mac?

I saw your magazine for the first time this month in a shop here in Sweden. As far as I know, it's the only one that addresses game development issues. I'm very interested by what I saw in the April/May 1997 issue. I would like to know if you cover Macintosh game programming, too. I'm very aware of how poor the Macintosh game programming world is when compared with what's available for PCs, so I would not be very

surprised if you did not devote too much space to it.

PEIRANO PHILIPPE
AXXORN, SWEDEN

EDITOR ALEX DUNNE RESPONDS: *Yes, the Macintosh market has dwindled to a secondary market for most game developers who still support it. If Apple can turn its platform around, you can be sure that you'll see coverage in GD. My heart goes out to Apple and all Mac users, but only time will tell whether games+Macintosh=developer \$\$.*

Who Are We?

I have seen somewhere that *Game Developer* is now a monthly magazine. Is this true? I haven't yet seen the April/May 1997 issue, so I don't know what changes have been made to the magazine; I hope that it continues its already great tradition of bringing timely information and articles on game development.

Another question: Who, in your personal opinion, are the main readers of GD? Also, how do you view the level of content, meaning, do you think that the articles are too technical or too introductory (but I guess this has to be balanced with what specific people *Game Developer* is targeting). Frankly, I wish *Game Developer* would become more technical and advanced-developer oriented. Then again, I guess it already is, and I would hate to have to read about triple-buffered sound layering (a term I just made up) without reading some introductory material first.

I guess it's quite hard juggling and compromising between content depth and breadth. Anyways, you guys are all doing a great job!

SANG WOO HAN
SOUTH KOREA

EDITOR ALEX DUNNE RESPONDS: *I'll look into what happened to your April/May issue. You should have received it by now!*

Regarding the balance between technical and introductory material, it definitely

is a challenge to hit the right level. I am trying to target people that develop games professionally, not beginners. However, since the professionals are generally so advanced that to keep them interested might take extremely technical articles (which then would only appeal to a small section of our readers — those who actually understand concepts like triple-buffered sound layering), I shoot for articles that are technical but can be understood by a wider audience. Of course, by being less technical, some readers won't be as interested.

Now I know how politicians feel when they try to please everyone — it just can't be done. You have to pick a path and stay on it.

New to Scene

Yesterday I read your magazine (April/May 1997 issue) for the first time — and I liked it very much. I have no previous magazines of yours to compare this issue with, but judging from that issue and the next one coming up, it's looking very good.

I especially like that designers, graphic artists, and sound engineers/musicians and the rest of the "supporting cast" get some very well deserved consideration.

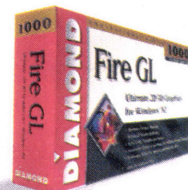
Often have I read things like, "The coders really made some cool graphics." I recognize that coders can have a big say in the quality of the graphics, but often graphic artists aren't mentioned at all. Also, the articles were good. "Designing an Effective Interface," "Determining Your Game's Audio Requirements," and the Artists View "Different Perspectives on 3D Sprites" were well written and documented, and even funny at times.

So, here's to the future! Keep up the good work!

Dennis W. Hansen
Via e-mail



FIRE GL 1000 WINDOWS NT ACCELERATOR. Now you don't need a workstation to create workstation-quality graphics. Diamond Multimedia's affordable new Fire GL 1000 brings the ultimate in 2D, 3D, and video acceleration to your Windows® NT desktop. It provides fast texture mapping and double-buffering—and supports the OpenGL, Direct 3D, and Heidi 3D APIs—for all your animation, authoring, and CAD applications. So visit us at www.diamondmm.com/firegl for all the facts and the location of your nearest Diamond reseller. Then just imagine what you can do.



Use your
imagination.

(Not a
high-priced
workstation.)



Visit us at www.diamondmm.com/firegl for a chance to win a Fire GL 1000 accelerator!



All trademarks are the property of their respective owners. ©1997 Diamond Multimedia Systems, Incorporated. All rights reserved.



Accelerate your world.

INDUSTRY WATCH

by Ben Sawyer

BANDAI HAS CALLED OFF the merger between itself and Sega; middle managers at Bandai supposedly rejected the idea on the grounds that it didn't make any sense. It's hard to figure out where Sega rebounds from here. Already Peter Loeb, one of the powers behind Heat, Segasoft's online gaming network, jumped to EA to head up its emerging online plans.

SPEAKING OF EA, the company has also hired two Virgin managers for Origin. Neil Yong and Chris Yates, two former V.P.s of technology, operations, and product development, left Virgin only to surface at Origin. Origin has had a lot of trouble lately losing key staff and is going through some restructuring, all while trying to get Ultima IX and the long-awaited Ultima Online out the door.

EA's news didn't stop at new executives either. On the heels of recent investments in both Mpath and Accolade, EA announced that it would absorb Maxis in a merger valued at \$125 million. The merger values EA at nearly \$1.8 billion. Reaction seems good given the forecast that EA will wring more out of the Sim product line on a worldwide basis than the current \$50 million in sales that Maxis has.

In other EA news, it announced, along with Microsoft, support for ICTV, a new set-top box technology that allows users to play online games run from remote PC servers over their cable TVs. This is a significant little piece of news. EA is especially hardcore about platforms they choose to support. It's rarely been wrong in its platform support, jumping on the Amiga, pulling out of 3DO, and propelling the Genesis to stardom. Could it be right again? Microsoft seems to think so. This development has people buzzing. Trials continue in San Diego.

IS THERE A SPIN-OFF in the midst? CUC announced a \$22 billion merger with HFS International. CUC, which owns Sierra, Davidson, Knowledge Adventure, Blizzard, Berkeley Systems, Papyrus, and others, has cobbled together a huge number of game companies. However, the CUC also has a

A Call to Microsoft

AN IMPRESSIVE LIST of game developers released an open letter to Microsoft Corp.

calling on the company to actively support the OpenGL 3D API for games on Windows 95 and Windows NT. The letter makes it clear that game developers want the option of using OpenGL, and they want Microsoft to work with them to provide OpenGL on its platforms. John Carmack, technical director of id Software, has made no secret of id's commitment to OpenGL. "The

best thing Microsoft could do for the game community would be to release the Windows 95 OpenGL MCD framework to allow hardware vendors to easily build robust, full featured drivers. Without Microsoft's help, there will be several partial implementations to satisfy specific requirements, resulting in version problems and incompatibilities. The strengths of OpenGL are important enough that it is going to be used one way or another, but life would be so much better if Microsoft cooperated."

We, the undersigned professional game developers, call on Microsoft to continue its active OpenGL development, to ship its DirectX build tools for OpenGL, and to implement the Windows 95 OpenGL MCD framework to allow hardware vendors to easily build robust, full featured drivers. We want our 3D API competition to happen on an open technical playing field, with us, the people who actually write the game, deciding which APIs we should and should not use. This open technical competition is healthy for the industry and will result in creating the technically competitive environment Microsoft must take part in creating the operating system, and we urge the company to be a positive force in doing so by actively supporting OpenGL. The entire PC game industry will benefit as a result.

Bill Buxton
Ken Brinkley
Samuel Blomberg
John Carmack
Glen Corpe
Steve Corpe
Mark DeBorja
Jim Dine
James Fleming
Rick Gentler
Ed Goldman
Chris Green
Robert Green
Mike Harrison
Chris Hecker
Lawrence Holland
Brian Hook
Andrew Howe
Brent Johnson
Rick Kermmer
Dennis Kersley
Donovan Leitch
John Lindner
Peter Lindner
Mike Luskovic
Jonathan Mayor
John Miles
Doug Murr
Cory Morrison
John Naylor
Mike Newhall
Jay Patel
John Romero
William Scahron
Jason Schall
Zachary Simpson
David Sifford
Tim Sweeney
Chris Taylor
Dave Taylor
Trey Taylor
Cameron Taler
Matt Toole
Neil Verheyle
Charles Walker
Kevin Wasserman
Patrick Wu
David Wu
Pat Wyatt
Billy Zelnick
Greg Zelnick

VideoSoft VSDirect

UK-BASED COMPONENTSOURCE has made available VSDirect, a set of ActiveX controls that facilitates the use of DirectX technology from within Visual Basic 4.0 and 5.0. An earlier version of VSDirect was previously supplied by MicroHelp under the name MicroHelp Game and Multimedia Xponents.

VSDirect contains numerous multimedia component features such as VSDirectDraw, VSDirectSound, and VSDirectPlay. VSDirectDraw works with interfaces that accelerate animation techniques through direct access to video memory and hardware. VSDirectSound allows hardware and software sound mixing and playback. VSDirectPlay enables connectivity of games over a modem link or network.

VSDirect is available for Windows 95 and Windows NT 4.0 or higher and is priced at approximately \$195 (£120), with specially priced multi-user site licenses available for corporate or developer teams. An upgrade of MicroHelp Game and Multimedia

Xponents to VSDirect is available for a special price of about \$30 (£20).

■ **ComponentSource**
Berkshire, United Kingdom
+44 (0) 118 958 1111
<http://www.componentsource.co.uk>

Media Manager

LUMINOUS TECHNOLOGY CORP., has announced the availability of the Luminous Media Manager content management system. This digital asset management system runs on industry standard Open Database Connectivity (ODBC) compliant databases that can be accessed from any operating platform, scaled to add processing power and system resources as needed, and extended with additional functionality from third-party tools. Using 14 customizable fields, Media Manager software customers can catalog and track file types including images, illustrations, digital sounds, QuickTime movies, text, and other digitized media over networks, the Internet, or enterprise-wide intranets.

Media Manager uses a plug-in architecture for its "I-Pieces," which generate thumbnails and previews of many

A S T S

O F G A M E D E V E L O P M E N T

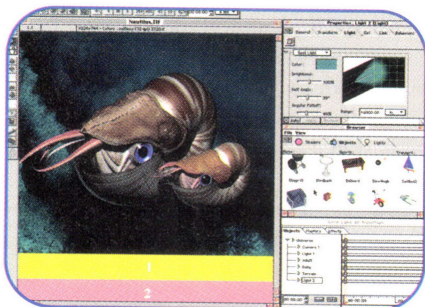
file types, including TIFF, PICT, EPSF, GIF, JPEG, QuickTime, and Mac OS sound files. Third-party developers can develop custom applications for current and future media formats with the Media Manager API.

Media Manager runs on MacOS or Windows servers. Users can access the database from MacOS, Windows 95, or Windows NT workstations, or through a web browser. A Media Manager Cross-Platform MacOS/Windows NT Server Kit is available for the introductory suggested retail price of \$3,495, which includes five client licenses. Additional client packages are available on the MacOS platform or Windows operating system for suggested retail prices of \$995 for five clients, \$1,795 for 10 clients, and \$3,795 for 25 clients. Site licenses are also available.

■ **Luminous Technology Corp.**
Seattle, Wash.
(206) 689-6309
<http://www.luminous.com>

Ray Dream Studio 5

METACREATIONS has announced the latest version of its 3D design and animation tool, Ray Dream Studio 5. This release features new object creation tools, new rendering effects, interface improvements, and support for MMX and symmetric multiprocessing (SMP) technology.



Vertex-level modeling has been implemented in the Mesh Form Modeler. ThinkFish LiveStyles lets users apply 2D hand-drawn looks to 3D objects, so images and animations rendered with the Natural Media renderer

can resemble cartoons, paintings, or sketches. The animation toolset has been expanded to include collision detection and other physically based behaviors. Plus, Ray Dream Studio 5 comes with over 850 Dream Models created by Acuris and Viewpoint Datalabs, 500 shaders and textures, and hundreds of preset camera and light configurations, as well as presets for behaviors, deformers, links, and render effects.

Ray Dream Studio 5 is available for MacOS, Windows 95, and Windows NT. The suggested retail price is \$449. Upgrade price is \$99.

■ **MetaCreations**
Scott's Valley, Calif.
(800) 846-0111
(408) 430-4000
<http://www.metacreations.com>

3D Game Machine

VIRTUALLY UNLIMITED has released version 2.0 of its 3D Game Machine framework for Windows 95, Windows NT, and MacOS. 3D Game Machine (3DGM) is a specialized game development tool, featuring a highly optimized core with a straightforward C/C++ API, as well as visual tools. Full integration with Kinetix 3D Studio MAX is provided through its 3DGM Factory interactive modeler, and the 3Dfx Voodoo is directly supported.

The 3DGM Factory lets artists to import animated 3D models for immediate integration into their game. Factory's groupware features organize the team's work for accurate asset tracking and automatic project updating as new 3D models and textures are added.

3D Game Machine 2.0 is licensed on a zero-royalty, per-application basis, priced at \$14,750 for Windows 95 and Windows NT, and \$7,750 for MacOS. An evaluation version of the 3DGM package is also available.

■ **Virtually Unlimited**
Geneva, Switzerland
(+4122) 700-7170
laurent@virtually3d.com
<http://www.virtually3d.com>

huge business running affinity programs and is especially big in the travel and services business. Now that it is merging with HFS, it is an even bigger travel and services company that includes Avis Car Rental, Days Inn, Howard Johnson, Coldwell Banker, Century 21, Ramada, and more.

Ken Williams, founder of Sierra, will be named vice chairman of the corporation — now we know the real meaning behind "King's Quest." While the market awaits the final details of the combined company, speculation is that the software companies will be spun off within a couple of years. Much will depend on how HFS stockholders and executives react. Other than Leisure Suit Larry becoming a new hotel mascot, there just doesn't seem to be much synergy here — as if there really was any.

FINALLY, BRODERBUND is trying to better define its commitment to the gaming market on which it was founded. The company has consolidated its gaming products, including Riven, the sequel to Myst, under its new Red Orb label. Red Orb also announced that it had signed Trilobyte to an exclusive distribution agreement. Joining Trilobyte and Cyan in producing games for Red Orb are SSG, Presto Studios, Smoking Car Productions, and Raven Software. Red Orb will develop in-house as well.

NEWS RESOURCE OF THE MONTH.

Are you using Pointcast (www.pointcast.com), Microsoft Investor (www.investor.com), or Mercury Mail (www.mercurymail.com) to track the stocks and news of public game companies? This month, I thought I'd include the symbols of all the relevant companies that I track.

AKLM	Acclaim Entertainment
ATVI	Activision
BROD	Broderbund
CU	CUC International
ERTS	Electronic Arts
GAME	Gametek
GTIS	GT Interactive Software
NTDOY	Nintendo
SEGNV	Sega Enterprises
SEVL	Seventh Level
SNE	Sony Corp.
SBYT	Spectrum Holobyte
SP	Spelling Entertainment (Virgin)
THDO	3DO Company

Multipass Rendering and the Magic of Alpha Blending

The 1997 Computer Game Developers Conference just came and went, and I'm typing this as I recuperate from its effects on my voice and health. This year's CGDC was pretty neat, and as always one of the highlights of the show was Michael Abrash's annual lecture. This year, he

talked about the architecture of id Software's *QUAKE*, and one of the techniques he described was GL*QUAKE*'s use of multipass rendering. So since it seems like this is going to be a hot topic for a while, I thought I'd explore the subject.

Multipass rendering is the technique of rendering a scene multiple times into the frame buffer, combining each rendering pass with the prior one to achieve interesting image composition effects. For example, you could multiply the passes together, or add them, or do a weighted blend on them, or whatever. What you do between passes is limited only by the breadth of your imagination, the speed of your hardware (multipass rendering's viability with a software renderer is doubtful, so I'm assuming hardware acceleration is present), and the flexibility of your API. Later on, I'll talk about specific effects that you can achieve with multipass rendering, but first we need to devise a way of combining our multiple passes together.

EQUATION 1. The alpha blending operation.

$$\text{final.color} = \text{fragment.color} * S_f + \text{pixel.color} * D_f$$

A quick side note: I'm not going to spend too much time worrying about software rendering or paletted rendering tricks, since that's "old tech." My columns will concentrate on advanced techniques that leverage hardware accelerators that use RGB rendering. So unless something freakish happens, it is highly unlikely that I will discuss techniques that don't rely upon hardware accelera-

tion, RGB frame buffers, and texture maps. Consider this fair warning.

A Crash Course on Alpha Blending

For multipass rendering to work, we need a way to combine, at each pixel, what is already in the frame buffer with the incoming color. This incoming color (sometimes referred to as the frag-

Multipass rendering is the technique of rendering a scene multiple times into the frame buffer, combining each rendering pass with the prior one to achieve interesting image composition effects.

ment color) is generated by applying texture mapping, lighting, filtering, and fog to your scene. Thus, the fragment is effectively the undithered color that we want to store into the frame buffer while rasterizing a primitive.

THEORY OF ALPHA BLENDING. Combining a fragment color with an existing pixel is such a useful operation that most major 3D graphics APIs support it in one form or another, usually using the moniker "alpha blending." Alpha blending combines a fragment color with the existing pixel color (Equation 1).

S_f and D_f are our source and destination factors, respectively;

fragment.color is the RGB color of our incoming fragment; and **pixel.color** is the RGB color of the pixel already in the frame buffer.

Tables 1 and 2 show the wide variety of possible source and destination factors. Note that these tables are not necessarily all encompassing — with OpenGL, for example, it's entirely possible that a vendor could implement new factors or even new blending equations (perhaps subtraction operations) through the OpenGL extension mechanism.

The mechanics of alpha blending is a lot to digest at once, especially since

EQUATION 2. Turning off blending.

$$\text{final.color} = \text{fragment.color}$$

EQUATION 3. Source factor of 1.0. Destination factor of 0.0.

$$\text{final.color} = \text{fragment.color} = \text{fragment.color} * 1 + \text{pixel.color} * 0$$

a lot of blending modes aren't particularly intuitive. Equation 2 shows a simple example of how to turn off blending mathematically by collapsing Equation 1.

Easy enough! We don't want the pixel color to contribute at all, and we want the fragment color to flow through unmodified. So a source factor of 1.0 and a destination factor of 0.0 should give us Equation 3.

Again, pretty simple. By using different source and destination factors, we can create a wide range of effects, including translucency, "dark maps," dark maps mixed with Gouraud shading, and specular lighting. But before getting into too much detail, let's go over how OpenGL and Direct3D support alpha blending.

ACCESSING BLENDING THROUGH OpenGL.

OpenGL's interface to alpha blending is embodied in only four functions: `glEnable`, `glDisable`, `glBlendFunc`, and `glTexEnv`. To enable and disable alpha blending, you call `glEnable` and `glDisable`, respectively, with the parameter `GL_BLEND`. Once you've enabled blending, you call `glBlendFunc` to specify your source and destination factors. Finally, `glTexEnv` specifies the method by which OpenGL will compute the source alpha factor when texture mapping is enabled (if texture mapping is not enabled, OpenGL will just use the alpha value iterated over the primitive's surface).

Since OpenGL specifies that alpha blending must be supported, you can just go ahead and use it without too much hassle. A caveat: If the underlying hardware doesn't support alpha blending or a particular set of blending factors, then an OpenGL implementation will fall back to a software-rendering implementation. If you're only rendering very few and/or very small alpha-blended polygons, then software rendering may not have a significant impact on performance. If you're doing multipass rendering, however, this fall back could be a killer.

ACCESSING BLENDING THROUGH DIRECT3D. Not surprisingly, Direct3D's mechanism for controlling blending bears a striking resemblance to OpenGL. Blending is enabled and disabled by setting the `D3DRENDERSTATE_ALPHABLENDENABLE` state token to `TRUE` and `FALSE`, respectively. The source and destination factors are controlled by the `D3DRENDERSTATE_SRCBLEND` and

TABLE 1. Source Factors.

OpenGL Constant	D3D Constant	Factor's Value
<code>GL_ZERO</code>	<code>D3DBLEND_ZERO</code>	0
<code>GL_ONE</code>	<code>D3DBLEND_ONE</code>	1
<code>GL_DST_COLOR</code>	<code>D3DBLEND_DESTCOLOR</code>	pixel.color
<code>GL_ONE_MINUS_DST_COLOR</code>	<code>D3DBLEND_INVDESTCOLOR</code>	1 - pixel.color
<code>GL_SRC_ALPHA</code>	<code>D3DBLEND_SRCALPHA</code>	fragment.alpha
<code>GL_ONE_MINUS_SRC_ALPHA</code>	<code>D3DBLEND_INVSRCALPHA</code>	1 - fragment.alpha
<code>GL_DST_ALPHA</code>	<code>D3DBLEND_DESTALPHA</code>	pixel.alpha*
<code>GL_ONE_MINUS_DST_ALPHA</code>	<code>D3DBLEND_INVDESTALPHA</code>	1 - pixel.alpha*
<code>GL_SRC_ALPHA_SATURATE</code>	<code>D3DBLEND_SRCALPHASAT</code>	min(fragment.alpha, 1 - pixel.alpha)*

*assumes the existence of an alpha channel in the frame buffer.

`D3DRENDERSTATE_DESTBLEND` state items.

Finally, the interaction of texture mapping and blending when computing source alpha is controlled by the `D3DRENDERSTATE_TEXTUREMAPBLEND` state token. Like OpenGL, Direct3D uses interpolated alpha when texture mapping is disabled (when `D3DRENDERSTATE_TEXTUREHANDLE` is `NULL`). Currently, the derivation of source alpha in certain cases (for example, `D3DBLEND_COPY` when no texture alpha is present) is inconsistent across shipping Direct3D drivers. When coding to Direct3D, be sure to keep handy as many hardware accelerators as possible so you can get a feel for how a certain feature is being interpreted by hardware driver writers (don't you just love this?).

Note that because Direct3D doesn't guarantee any functionality from a particular driver, you must verify the availability of alpha blending and the specific blending factors you intend to use; otherwise, the output generated may be unexpected. Unlike OpenGL, if some effect — such as alpha blending — isn't supported by a driver in Direct3D, you'll be forced to either eschew that feature altogether or implement it yourself; neither option is particularly enticing.

Cool Effects

Okay, here's the meaty part: cool things that you can do with multipass rendering. I'm just touching on some of the effects that I've been exposed to over the years. I'm sure developers will come up with some pretty wacky uses for blending that I can't even comprehend right now, but this should get you started.

TRANSLUCENCY. When developers hear the phrase "alpha blending," the first effect that usually pops to mind is translucency — the ability to see partially through a surface. Translucency has many obvious uses.

- Rendering glass, plastic, and other translucent materials
- Glass reflections
- Heads-up displays (HUDs)
- Logos, symbols, or text overlays on the screen for things like copyright and disclaimer information
- Map overlays
- Lens-flare effects
- Shield effects
- Water

We control a surface's translucency/opacity by using the fragment's alpha value as a blending factor (Equation 4).

TABLE 2. Destination Factors.

OpenGL Constant	D3D Constant	Factor's Value
<code>GL_ZERO</code>	<code>D3DBLEND_ZERO</code>	0
<code>GL_ONE</code>	<code>D3DBLEND_ONE</code>	1
<code>GL_SRC_COLOR</code>	<code>D3DBLEND_SRCCOLOR</code>	fragment.color
<code>GL_ONE_MINUS_SRC_COLOR</code>	<code>D3DBLEND_INVSRCCOLOR</code>	1 - fragment.color
<code>GL_SRC_ALPHA</code>	<code>D3DBLEND_SRCALPHA</code>	fragment.alpha
<code>GL_ONE_MINUS_SRC_ALPHA</code>	<code>D3DBLEND_INVSRCALPHA</code>	1 - fragment.alpha
<code>GL_DST_ALPHA</code>	<code>D3DBLEND_DESTALPHA</code>	pixel.alpha*
<code>GL_ONE_MINUS_DST_ALPHA</code>	<code>D3DBLEND_INVDESTALPHA</code>	1 - pixel.alpha*

*assumes the existence of an alpha channel in the frame buffer.

EQUATION 4. Using fragment's alpha value as a blending factor.

$$\text{final.color} = \text{fragment.color} * a + \text{pixel.color} * (1 - a)$$

LISTING 1. OpenGL version.

```
glEnable( GL_BLEND );
glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
glTexEnv( GL_MODULATE );
```

LISTING 2. Direct3D version.

```
pDev->SetRenderState( D3DRENDERSTATE_ALPHABLENDENABLE, TRUE );
pDev->SetRenderState( D3DRENDERSTATE_SRCBLEND, D3DBLEND_SRCALPHA );
pDev->SetRenderState( D3DRENDERSTATE_DESTBLEND, D3DBLEND_INVSRCALPHA );
pDev->SetRenderState( D3DRENDERSTATE_TEXTUREMAPBLEND, D3DTBLEND_MODULATEALPHA );
```

The source alpha component (*a*) controls the weighting between the pixel and fragment colors. Using the alpha blending configuration from Equation 4, as source alpha varies from 1.0 to 0.0, a surface will become gradually more transparent.

The OpenGL code in Listing 1 will enable blending for translucency effects, and will compute source alpha as the product of vertex and texture alpha (note that if there is no alpha in the texture, then texture alpha will implicitly be 1.0). An important point is that both Direct3D and OpenGL use the term “modulate” to mean “multiply.” I don’t know why this is, other than “modulate” sounds cooler than “multiply,” but you should be aware of this peculiar terminology.

Listing 2 is the Direct3D DrawPrimitive analog of the previous OpenGL code. (From now on, I’ll be using the DirectX 5 DrawPrimitive-style Direct3D code snippets, mostly because execute buffers are absolutely awful for doing code fragments.)

To achieve correct output, an application has to render translucent objects in back-to-front order. Otherwise, nearer translucent surfaces won’t correctly blend with more distant surfaces.

DARK MAPS. One of the coolest uses of alpha blending is for multipass dark maps. A dark map, commonly referred to as a light map (I’ll explain why I use the term “dark map” later), is a texture map used to represent areas of shadow and light cast on a surface, independent of the underlying surface material’s base texture. Dark maps

allow for far more striking lighting than the more traditional per-vertex illumination models (such as Gouraud shading), where colors are only linearly interpolated across the face of a polygon.

Dark maps are great because they allow gorgeous lighting effects without resorting to prelighting textures (which consumes gobs of memory) or an unwieldy surface cache architecture (à la the software-only version of *QUAKE*). And because dark maps can be stored as monochrome textures of fairly coarse granularity, they can be modified and downloaded quickly for dynamic lighting without interfering with the base textures. Thus, higher resolution base textures can stay resident in texture RAM.

COLORED DARK MAPS USING RGB TEXTURES.

Dark maps are implemented using two-pass rendering: the first pass renders the dark maps, and the second pass renders the base textures. For colored lighting effects, you’ll usually want to use RGB dark maps; during the second pass, the alpha blender is configured to

multiply the fragment color with the existing color (Equation 5). The OpenGL form of Equation 5 is in Listing 3. The Direct3D form is in Listing 4.

This blending mode assumes that your incoming dark texture map is in RGB format, or at least in a format such that the fetched texel will be converted to RGB before blending. For example, OpenGL supports two monochrome formats, *GL_INTENSITY* and *GL_LUMINANCE*, which should consume less memory than an RGB texture format because they support only gray scales.

MONOCHROME DARK MAPS USING ALPHA

TEXTURES. Unfortunately, some of the poorer hardware accelerators out there don’t support “destination color” blending. They support only “source alpha” blending. When confronted with this sort of degenerate hardware, you can use an alpha-only texture and configure your blender such that you ignore the fragment color (source factor is *GL_ZERO* or *D3DBLEND_ZERO*) and multiply the frame buffer contents by source alpha (destination factor is *GL_SRCALPHA* or *D3DBLEND_SRCALPHA*). Equation 6 shows what this workaround looks like mathematically.

Because an alpha-only texture possesses no color content, you’re going to be stuck with monochromatic gray-scale shadows, albeit with the benefits of a reduced memory footprint and better support for hardware with limited alpha-blending support.

MIXING DARK MAPS AND VERTEX LIGHTING. At first glance, dark maps and vertex-oriented lighting — such as Gouraud

LISTING 3. OpenGL version.

```
glEnable( GL_BLEND );
glTexEnv( GL_REPLACE );
glBlendFunc( GL_DST_COLOR, GL_ZERO );
```

LISTING 4. Direct3D version.

```
pDev->SetRenderState( D3DRENDERSTATE_ALPHABLENDENABLE, TRUE );
pDev->SetRenderState( D3DRENDERSTATE_SRCBLEND, D3DBLEND_DEST_COLOR );
pDev->SetRenderState( D3DRENDERSTATE_DESTBLEND, D3DBLEND_ZERO );
pDev->SetRenderState( D3DRENDERSTATE_TEXTUREMAPBLEND, D3DTBLEND_COPY );
```

EQUATION 5. Multiplying the fragment color by existing color.

$$\text{final.color} = \text{fragment.color} * \text{pixel.color} + \text{pixel.color} * o$$

THE LATEST MOVIE

superhero

AND HIS sidekick.

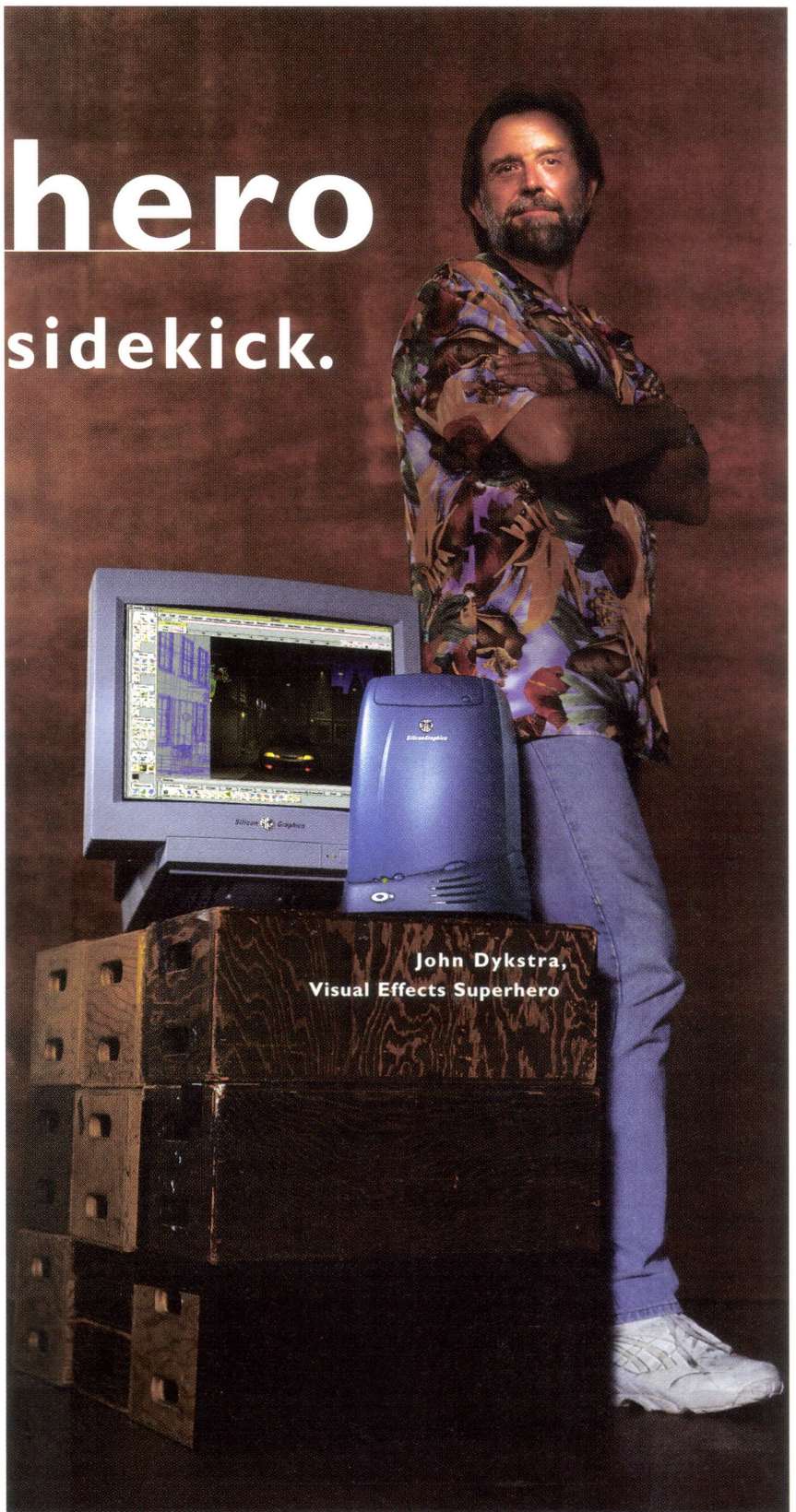
It takes a man with larger-than-life power to create the visual effects that make Batman™ & Robin™ perform their death-defying feats. And while JOHN DYKSTRA makes that look easy, he always has a budget looming overhead as well. That's why John counts on Silicon Graphics solutions like the O2™ workstation. "Their high-performance platforms have been instrumental in helping us work more efficiently and economically. No longer can you say to the director 'that's impossible, you can't do it!' Because we can! We can do more iterations of each shot than ever before and still stay within budget." And O2 makes it all possible. See our Website or call 800.636.8184, Dept. LS0086 for more information. And get yourself the system that'll create special effects for your budget.

O2

DESKTOP WORKSTATION

\$7,495

www.sgi.com/O2



**John Dykstra,
Visual Effects Superhero**

See what's possible •



SiliconGraphics
Computer Systems



© 1997 Silicon Graphics, Inc. All rights reserved. Silicon Graphics and the Silicon Graphics logo are registered trademarks, and O2 and See what's possible are trademarks, of Silicon Graphics, Inc. MIPS and the MIPS RISC Certified Power logo are registered trademarks, and R10000 and R5000 are trademarks, of MIPS Technologies, Inc. BATMAN, ROBIN, and all related characters, names and indicia, are trademarks of DC Comics © 1997.

See us at SIGGRAPH booth #1139 & #1339

LISTING 5. Setting OpenGL texture environment to GL_ADD.

```
glTexEnv( GL_ADD )
```

shading — seem mutually exclusive. This isn't necessarily true. Vertex colors can be used for two powerful effects in conjunction with dark maps — dynamic lighting without dark map recomputation, and rendering colored dark maps even when those maps are monochromatic.

Dark maps only modulate the color of the underlying texture. They are not capable of brightening a scene, only darkening portions of it. (This is why the term “dark map” is technically more accurate than “light map.” I'd use the term “shadow map,” but that's already used to describe something else entirely.) For this reason, introducing dynamic lights to brighten a scene typically requires recomputing the dark maps, à la GLQUAKE, which can be fairly expensive computationally and affect of texture download performance.

However, vertex lighting can brighten a scene using dark maps without imposing undue overhead, if you're willing to accept the limitations of vertex lighting (these being a lack of perspective correction and the need for heavy tessellation to render highlights well). Instead of rendering the dark maps using the API's “replace/copy” texture environment, add the interpolated color to the dark map color. With OpenGL, this is accomplished by setting the texture environment to GL_ADD (Listing 5). Unfortunately, the GL_ADD texture environment isn't a standard part of the OpenGL API; however, several OpenGL vendors are considering it as an extension. With Direct3D, you add the interpolated color to the dark map color by setting the texture blend mode to D3DTBLEND_ADD (Listing 6). Now,

EQUATION 6. Destination color workaround.

$$\text{final.color} = \text{fragment.color} * o + \text{pixel.color} * \text{fragment.alpha}$$

LISTING 6. Setting Direct3D texture blend mode to D3DTBLEND_ADD.

```
pDev->SetRenderState( D3DRENDERSTATE_TEXTUREBLEND, D3DTBLEND_ADD )
```

whenever a dark map is rendered, it will have the vertex colors added to it, allowing dark maps to be brightened.

Earlier, I mentioned that one of the drawbacks of using an intensity or alpha format for dark maps is that this limits a game to using gray-scale dark maps. Once again, vertex lighting can solve this problem by rendering the dark maps using the API's modulate texture environment, multiplying the color of the dark map with the color values interpolated from the vertices. These vertex colors will likely be precomputed at the same time the dark maps are generated (probably by using some external tool). With OpenGL, the texture environment must be set to GL_MODULATE; with Direct3D, the texture blend mode must be set to D3DTBLEND_MODULATE. These two techniques are mutually exclusive on a single pass, because they both need control of the texture environment and vertex colors.

DYNAMIC LIGHTING WITHOUT USING VERTEX COLORS OR RECOMPUTING DARK MAPS. It would be nice if there was some way to dynamically light a scene without using vertex lighting or recomputing dark maps. And believe it or not, such a technique actually exists, although it requires a third pass — bear with me, it's not as bad as it sounds (usually). The basic technique is simple — render your dark maps normally as part of your first pass. Now it gets tricky; on the second pass, render those parts of the scenes affected by dynamic lights,

and add the dynamic lights into the frame buffer. How you render the dynamic lights is up to you, because there are quite a few ways of doing this (heavily tessellated Gouraud shaded polygons, using “brightening” texture maps, and so on). Your alpha blender needs to be configured as in Equation 7. The OpenGL code is in Listing 7. The Direct3D version is in Listing 8.

This will brighten the dark maps without recomputing or reuploading new dark maps, and also without using vertex lighting. The final pass is rendered using the standard multiplicative dark map blending mode.

The “brightening” phase can be considered a “half pass,” occurring between the dark map and texturing passes. The sole purpose of this pass is to brighten up those regions of the scene specifically affected by dynamic lights. If the portion of the scene being illuminated by dynamic lights isn't very large, then the performance hit for this partial pass may not overwhleming, and may actually be less than the hit taken for recomputing and reuploading dark maps.

In the future, both OpenGL and Direct3D will provide multitexture extensions. Multitexture extensions will allow developers to specify multiple textures per face and multiple texture coordinates per vertex and to dictate how the multiple textures combine. Thus, a combined dark map/texture map can be rendered in a single pass by the application (even though an underlying API or driver implementation may render it in two or more passes). Such extensions will allow hardware to implement dark maps and related effects without

EQUATION 7. Configuring alpha blender for dynamic lighting.

$$\text{final.color} = \text{fragment.color} * i_1 + \text{pixel.color} * i_1$$

LISTING 7. OpenGL version.

```
glBlendFunc( GL_ONE, GL_ONE );
```

LISTING 8. Direct3D version.

```
pDev->SetRenderState( D3DRENDERSTATE_SRCBLEND, D3DBLEND_ONE );
pDev->SetRenderState( D3DRENDERSTATE_DESTBLEND, D3DBLEND_ONE );
```

Develop your gameplay,
not your game engine.

Realism NETIMMERSE

automatically scales the visual realism of your 3D scene based on the computer's resources: from standard PC's to MMX to 3D graphics hardware with Direct3D, OpenGL or our proprietary software renderer.

Realtime NETIMMERSE

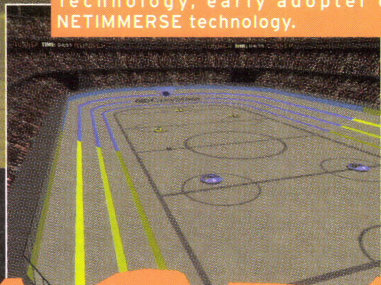
minimizes the work your application must do by culling out polygons not directly seen by the player: Visual Occlusion Culling, LOD, BSP's, portals... So your game looks better and runs faster than your competition.

Real Fast

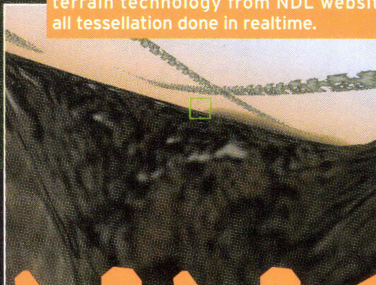
Get your title in stores before your competition with the features and tools you need: 3D Spatialized Sound, Collision Detection, Video Textures, Behavior, Optional Terrain Rendering Module, Imports 3DStudio™, OpenFlight™, VRML.

FULL SOURCE CODE, NO ROYALTIES

MagBall® arcade game by GreyStone Technology, early adopter of NETIMMERSE technology.



Defend3D: Free demo game of morphing terrain technology from NDL website, all tessellation done in realtime.



Realtime snapshot from "iF-22 Raptor" by Interactive Magic: 80,000 sq. miles of morphing terrain using NDL technology.



NETIMMERSE

C++ Game Engine

for creating Scale-able 3D PC Content

NUMERICAL DESIGN LTD.

www.ndl.com sales@ndl.com

West Coast +1 (415) 328-4388 East Coast +1 (919) 929-2917

Canyon Runner image courtesy of GreyStone Technology. ©1997 NUMERICAL DESIGN LTD.



LISTING 9. Specular highlights with OpenGL.

```
glTexEnv( GL_MODULATE );
```

requiring multiple passes, potentially improving performance dramatically.

TWO-PASS SPECULAR LIGHTING. Another great use of multipass rendering is to integrate specular lighting into your texture-mapped scene. Specular lighting is the additional lighting component that results from the reflection of light directly back to the viewer. Specular lighting is an additive component in most lighting equations and thus can saturate colors, unlike pure diffuse lighting, which only modulates colors.

With Direct3D, the specular color is iterated separately and is specified in the `D3D(T)LVERTEX::specular` member. In theory, you can have specularly lit (at



A scene from *QUAKE* with dark maps applied.

the vertices) textures in a single pass, assuming that the driver has implemented this feature. With OpenGL, all lighting is evaluated into a single "diffuse/specular" vertex color, so if you wish to add specular lighting to a diffuse-lit textured scene, you'll have to use a second pass. Extensions to OpenGL are being proposed to address this deficit.

To generate specular highlights with Direct3D (when there is no support for specular lighting) or OpenGL, you render the scene normally on the first pass: that is, with diffuse-lit textures. The OpenGL code is in Listing 9. The Direct3D version is in Listing 10.

During the second pass, the scene is rendered sans texturing, with specular lighting enabled (you don't want to reapply the diffuse lighting) and Z-buffer writes disabled (this isn't absolutely necessary, but is probably a

LISTING 10. Specular highlights with Direct3D.

```
pDev->SetRenderState( D3DRENDERSTATE_TEXTUREMAPBLEND, D3DBLEND_MODULATE)
```

LISTING 11. OpenGL "night vision" effect.

```
glEnable( GL_BLEND );
glBlendFunc( GL_DST_COLOR, GL_ZERO );
// set vertex colors to (0.0F,1.0F,0.0F)
// and draw a screen size triangle
```

performance win on a lot of accelerators). You then configure the alpha blender for additive blending using the same method described for the "two-and-a-half pass" light-map updates.

The second pass will add specular highlights at vertices to the scene. To improve performance, an application should render only those portions of the scene that have specular highlights.

FAKING OLD-FASHIONED PALETTE EFFECTS. I feel obligated to mention this, even though I hate it. Back in the Good Old Days of VGA, you could do a lot of neat tricks with palette manipulation. But the move to true RGB has forced a lot of programmers to reevaluate some of these hacks. Two of the more common effects that programmers used palette tricks for were full-screen color fades (for example, firing the fusion cannon in Parallax Software's *DESCENT*) and night-vision effects (everything is green).

Alpha blending can sort of emulate these tricks by rendering an screen-sized alpha blended polygon after you've rendered the rest of the scene. All together now...eeeeewwwwww. Yes, it's horrible. The obvious reason that I hate this hack is that you eat an entire level of depth complexity (and suffer the associated performance hit) for no other reason than to fake some cheesy effect.

For a full-on fade to some color à la the *DESCENT* fusion cannon, a huge translucent polygon works fine — vary the alpha value over time to fade from transparent to full opaque. It would be nice to figure out a way to use fog for this effect instead, but both OpenGL and Direct3D have pretty limited fogging mechanisms for accomplishing this type of effect.

For the night-vision effect, a huge polygon that acts as a green filter (attenuate out red and blue, pass green through unchanged) works adequately

to fake the effect (note that real night-vision systems are absolutely nothing like this). The OpenGL code is in Listing 11. (Technically, you could accomplish the same effect with OpenGL by enabling only writes to the green channel using `glColorMask`, but that's a pretty limited technique, and it's not available under Direct3D).

Remember, these are hacks, pure and simple, and not meant to be cutting-edge programming techniques. If someone out there comes up with better techniques for achieving these effects, please write me and I'll pass it along. In a forthcoming column, I'll talk about some ways to achieve these affects using hacks that don't rely on alpha blending.

Performance Implications

An application that uses multipass rendering will be stressing the triangle throughput and fill-rate capabilities of the API and hardware, assuming that each pass completely rerenders the entire scene. Not much can be done about the fill-rate hit, but an application can try to minimize the transformation overhead if it's willing to do all the transformations itself and use the API only for rasterization.

If this is the case, a game can transform, project, and clip the scene once, but at each triangle render it twice (for example, once with a dark map and then with the base texture). This approach will reduce the amount of transformations performed, but at the cost of eating a state change at each tri-



The same *QUAKE* scene in "fullbright" mode, with no dark maps applied.

angle (blending mode, blending function, and current texture map values will be changed at least twice per triangle). With some hardware, these state changes can be so costly that it actually makes sense to just retransform the whole scene so that changes to the state are minimized as much as possible. Alternatively, an application can cache the transformed vertex data and attempt to render triangles by material order, but the cache effects of this may be pretty grim.

As I stated earlier, at this time neither OpenGL nor Direct3D support rendering triangles with multiple simultaneous textures. However, such enhancements are likely in the future, making much of this maneuvering moot — you simply select the texture map, dark map, and a “mixing” function for each triangle, and hopefully, the API and/or driver will just do the right thing.

Another — hopefully short-lived — performance drawback of multipass rendering is that enabling alpha blending can introduce a performance penalty on many hardware architectures (due to the overhead of reading from the frame buffer). The effect of this performance hit can be negligible to catastrophic, depending on the hardware accelerator.

Quality Implications

Another drawback to multipass rendering is the limited precision of a typical frame buffer. When you're using a 16-bit display and rendering to it many times, you will suffer a loss of precision due to quantization error. This error will manifest itself as visible banding when too many passes are rendered.

No clear solution to these artifacts exists, aside from increasing the resolution of the frame buffer. A 24-bit frame buffer is far less susceptible to multipass banding than a 16-bit frame buffer, but it may be a while before we see 24-bit frame buffers in widespread use for games.

Caveats

Of course, nothing's ever quite as simple as we'd like, so I'll talk briefly about some of the pitfalls you may encounter when integrating multipass rendering into your game.

ORDER OF OPERATIONS. With multipass rendering it's possible to go absolutely crazy and render a scene using a combination of just about every effect imaginable, with the caveat that performance will likely suffer if you go overboard. I've presented a mixed bag of techniques that can be combined in many different ways; always keep in mind your desired order of operations! When mucking about with texture environments, alpha-blending modes, and multiple passes, it's easy to accidentally render things out of order, particularly when dark maps are involved. For clarity, you should write down exactly what you're trying to accomplish, then go through your passes and make sure that what you're getting is what you want.

Z-BUFFERING. Keep in mind that if you're Z-buffering, you must use the “less or equal” (`GL_LEQUAL` or `D3DCMP_LESSEQUAL`) comparison function for the second pass to appear. This mistake is easy to make, since programmers often will use `GL_LESS` or `D3DCMP_LESS` by default.

COMPATIBILITY. When it comes to compatibility, alpha blending suffers from one big drawback — only a handful of current accelerators support full alpha blending. Vendors had many understandable reasons to ditch alpha blending in early hardware designs, including cost, performance, and lack of encouragement from software developers. My theory is that alpha blending didn't become a priority until some weenie product manager needed to fill in one more bullet item for a presentation and randomly selected a feature from a graphics textbook... but that's just me being optimistic (as if product managers own graphics textbooks). To make matters worse, Microsoft's Direct3D team placed a strong emphasis on chromakey and stippled transparency instead of true alpha blending (as a matter of fact, alpha blending was/is simply broken in Direct3D Retained Mode), mostly because they were obsessed with software rendering performance and didn't seem to understand that alpha blending served a larger purpose than just translucency.

For these reasons, there is a large installed base of accelerators that don't support alpha blending. If you want to be bold and say that this sizable group

of useless accelerators is irrelevant, go for it and surf the cutting edge. For those of you targeting every accelerator in existence, you may not want to emphasize alpha blending too much because of its current lack of support. I'm fairly certain that all the next generation of accelerators (early to mid 1998) will have support for alpha blending — those that don't, well, you can just assume they aren't going to matter.

If you want to mess around with this stuff today, I'd recommend that you pick up either a Rendition Verite or a 3Dfx Interactive Voodoo accelerator (or both, since they can coexist in a system); these two chipsets seem to implement the most important alpha blending capabilities correctly. Hopefully, by the time this article is in your hands, some newer hardware from other manufacturers will be shipping with comprehensive alpha-blending support.

Fog. One downside to multipass rendering is that fog doesn't “just work,” at least not with OpenGL. OpenGL specifies that fogging occurs before alpha blending, which makes multipass rendering with fog significantly more complicated than it could be. There are ways around this, and in a future column, I plan on going over some of the subtleties of fogging with multipass rendering.

With Direct3D, fogging is even worse — it may or may not be broken, depending on how a particular hardware implementation is done. My best advice is to avoid using fog with multipass rendering with Direct3D — it's tough enough getting basic output from Direct3D — trying to do multipass with fog with Direct3D is just begging for trouble. ■

I'd like to thank Michael Gold and Gary Tarolli for reviewing this month's column.

Brian Hook is a poser who somehow managed to land a job at id software. When he's not coding or playing with his dog he manages to write the occasional column for Game Developer.

FOR FURTHER INFO

An excellent reference on the subject of using texture maps for shadows and lighting effects is *Fast Shadows and Lighting Effects using Texture Mapping* by Segal, Korobkin, van Widenfelt, Foran, and Haerberli (SIGGRAPH '92).

serious 3D finally comes to gaming

PERMEDIA

GLINT

3Dlabs

3Dlabs is the chip company that started the 3D revolution on the PC. 3Dlabs' GLINT processor has enabled great development tools such as 3D Studio MAX, Lightwave, Softimage and GameGen to run amazingly fast on the PC.

3Dlabs' PERMEDIA is now bringing the same revolution to game delivery.

PERMEDIA gives you robust, sizzling Direct3D, OpenGL and QuickDraw 3D acceleration, blazing Winmarks, awesome MPEG acceleration and superb VGA legacy support - all integrated into one cool package.

No longer must you decide between having great game performance and being able to run everything else out there. You can even run your favorite authoring tool on a low-cost PERMEDIA board.

What *is* the world coming to?

3D that enhances your PC and enhances your games - that's what. Check out 3D from the people who understand about developing *and* delivering great games.

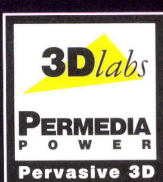
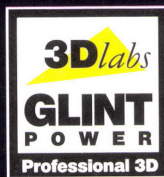
*For a full listing of boards and systems using PERMEDIA and GLINT
visit our web site or call:*

<http://www.3dlabs.com>

Telephone: (408) 436 3455

3Dlabs is working closely with hardware and software developers to get high-performance 3D into the hands of the games community.

Accelerator boards using our chips carry the 3Dlabs Power Logos as a guarantee of high performance and quality.



Developers wanting the inside track on all the latest cool stuff from 3Dlabs should register on our developer program by emailing developer.relations@3dlabs.com - or visit our web site at www.3dlabs.com

Take Me Into the Ball Game

It's about providing experiences. One of the enduring attractions of computer games is their ability to take us to fantastic places and immerse us in adventures far outside the realm of our daily existence. How else could you hope to get up-close and personal with the Orcs of Azeroth, the battlemechs of clan Jade Falcon, or... the Green Monster of Fenway? That's right: Game artists know that it's plenty tough to concoct a fantasy or sci-fi milieu from scratch, but a very different set of challenges

awaits those daring or foolhardy enough to recreate recognizable real-world settings for their players' enjoyment.

Sure, it may sound easier at first, and in some ways it is. The convenient thing about the real world is, it exists. The artist doesn't have to make it up. But there's a dark flip-side to that coin.

"The problem with a recognizable locations and characters," admits Andrew Johnston of Wizbang! Software Productions, "is that people know exactly what they are looking at and can quickly discern any differences between the game and the real world." Johnston is programming lead and project manager at Wizbang! on the MICROSOFT BASEBALL 3D project. The ambitious goal of this game-in-progress (due in stores November 1997) is not just to create an entertaining 3D baseball game, but to give fans of the national pastime the opportunity to go to bat as their favorite slugger in the hallowed arena of their hometown stadium, to play on big-league ballfields we mortals only get to see from the stands or on TV, and to rub elbows with recognizable hardball heroes. Moreover, the objective is to provide this experience in a more convincing manner than has been accomplished by previous baseball games: with ballplayers, fans will recognize without having to see a name in the corner of the screen and with authentically detailed ballparks, not to mention a real-world physics model, ambient 16-bit sound, and force-feedback joystick control that lets you feel the ball connecting with your virtual bat and the difference between running on outfield grass or baseline dirt. A look at what Microsoft and Wizbang! have gotten

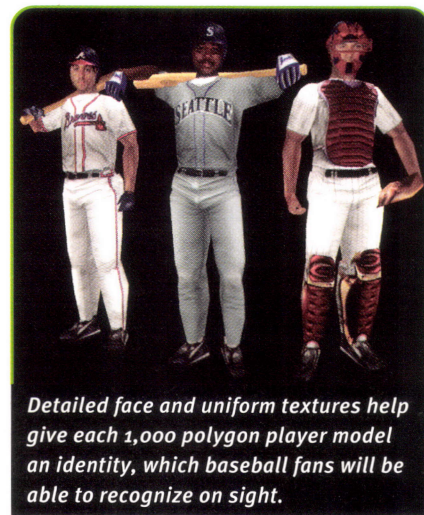
themselves into can be eye-opening for artists and game developers who expend their creative efforts bringing players face-to-face with creatures that never lived in settings that don't exist.

In the Ballpark

Though Wizbang! is developing the game, the challenge of capturing the essence of Major League Baseball began at the Microsoft Games Group, where most of the content — such as 3D models and textures — is created. Microsoft began by contracting with model-making mercenaries 3Name3D to create most of the prototype models for the 28 Major League Baseball stadi-

A real-world environment and characters may sound easier than concocting a fantasy world from the ground up, but listen to what happens when you set out to create the most realistic 3D baseball game yet.

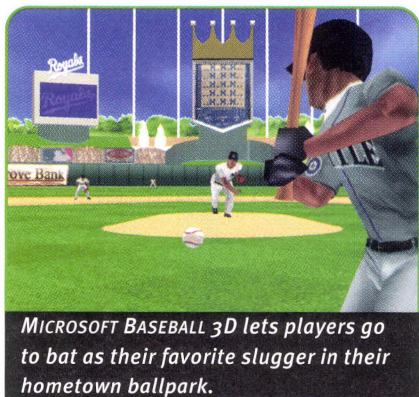
ums. The first park — Baltimore's Oriole Park at Camden Yards — came in at about 28,000 polygons. Before the model's delivery, however, the polygon budget had been drastically reduced. Artists at Microsoft went to work trying to make the model compatible with the hardware-accelerated target user system and with their game engine, Wizbang!'s ADLIB (Authoring and Design Language for Interactive Behaviors). (See *Game Developer*, August/September 1996, "Bringing Life to Hyperblade" for more information on ADLIB.)



Detailed face and uniform textures help give each 1,000 polygon player model an identity, which baseball fans will be able to recognize on sight.

At this stage in the project, the 3D accelerator boards Microsoft was developing for were still in the beta stage of development themselves. Given the inchoate nature of the target hardware, determining accurate budgets for polygons and texture maps was difficult. Jim Millar, 3D art coordinator at Microsoft, described the process as "trial and error, with a bit of guessing thrown in."

"We wanted authenticity with each stadium," asserts Tracy Curtis, who worked with Millar to recreate the Camden Yards model to fit within the



MICROSOFT BASEBALL 3D lets players go to bat as their favorite slugger in their hometown ballpark.

parameters of the game engine and target system. "We want the player to recognize the stadium he or she has chosen to play in."

Still, though *BASEBALL 3D* would be reliant on hardware acceleration and thus less constrained in terms of the amount of detail it could afford, a very real limit still had to be observed for stadiums' polygon count and texture memory budget. The sought-after realism would have to come from ingenuity and finesse rather than brute-force accretion of detail. To complicate matters, the game features a player-controlled camera offering unlimited viewing angles, meaning that each stadium has to be fully realized. Everything that could be seen from the field *would* be seen.

Armed with stadium blueprints, photos, illustrations, and photographs of architectural models, Millar and Curtis diligently reworked the Camden Yards model an estimated 30 times before getting what they wanted. They had scaled the original 28,000 polygons down to only 850, with a texture memory budget of 350 - 800K, depending on the user's video board capabilities. With placeholder textures applied, the working stadium model could be handed off to programmers while Millar and Curtis tackled several additional models that had been built to the original high-resolution specifications.

With most of the geometry problems resolved, Millar and Curtis turned to refining textures. Their goal was to compensate for the coarsening of detail that resulted from the simplified geometry and to capture the details that bring a stadium to life. In addition to official stadium reference materials, they looked elsewhere to get the details right.

"We've gone outside (and discovered there is an outside to Microsoft!) and

taken pictures of grass and trees. We look at postcards of city skylines," Millar explains. "We ask our friends around the country to take pictures of stadiums for us when they go to games. I even had my grandmother-in-law, who's a librarian in Baltimore, fax me a picture of the Baltimore city flag."

This attention to detail must span all 28 Major League Baseball stadiums, including different textures for night games. Though many textures can be shared between ballparks, each park has at least as many custom textures as shared textures. The aim is to give each venue a character all its own, recognizable by local fans and rivals alike.

"If it isn't close to the real thing, then the game wasn't worth making," Curtis explains of the effort. "There are some great baseball games that do have authentic stadiums, but lack the unique texture details our game provides."

Men in Uniform

The real stars of the game, of course, whether onfield or onscreen, are the ballplayers. As much as detailed stadiums add to the game, it's the fabled Boys of Summer who give it life, but only to the extent that artists and developers are able to capture the essence of that life. That objective suffered a hit early on, as the decision was made to keep the *BASEBALL 3D* ballplayer model relatively low-resolution due to budget and time constraints.

"The game is made to be scalable," explains *Wizbang!*'s Andrew Johnston, "so that *BASEBALL 3D* will use all of the features that a high-end board supports, while also running on low-end boards that may not support all features. On earlier projects, the costs of generating both low-polygon models for the software-rendered version of a game and high-polygon models for the hardware-accelerated version were prohibitive. In this case, the decision invariably was not to build the higher-detailed models, since low-polygon models could be used for all boards. This is a big disservice to those 3D accelerators that can handle much higher throughput rates."

"The challenge is making an organic form — the human figure — look realistic when built out of flat, angular polygons," points out Microsoft Games Group art director Mary Jo Kovarik.

Although at 1,000 polygons, the "low-polygon" ballplayer model allows more detail than many figures in recent 3D games, it was a bit of a come-down from the original 2,000 polygon prototype, described by modeler Randy Reeves as "quite dashing." The tight polygon budget adds an extra challenge to Microsoft's effort to create recognizable athletes, part of which involves matching models to motion-capture data. The upshot is that artists have had to work harder to breathe life into the still rather blocky 1,000 polygon model. Make that models: separate versions for fielders, batters, and catchers, as well as six increasingly lower-polygon LODs for each, the smallest of which allows only 28 polygons.

"Great attention had to be paid to make the physique of the modeled character match up with the skeleton from the motion-capture session," Kovarik explains. "Several revisions were necessary to get the model working properly with the game engine."

"It has been quite a delicate balancing act," Reeves allows with stoic understatement, "keeping the polygon count within tolerance, keeping polygons planar, and still accept motion data, still have decent-looking players, still remain generic enough."

"Generic" because a single, all-encompassing body-type has been settled on. The initial plan was to have several body types to accommodate different physiques: heavyset, tall, thin, and so on. Time — the enemy of all game developers — did away with that aspiration as well.

Yet Microsoft's artists and *Wizbang!*'s developers remain determined to capture the identities and personalities of players. Multiple instances of the generic model are created within the game for each of the players on the field. This allows each ballplayer to be individually customized with the correct skin color, uniform, and jersey details. Further, a lot of polygons — 13% of the total — are used in the ballplayer model's head to add realism. A range of detailed generic face textures provide even more character. Unique face textures have been created to represent the most famous and recognizable big leaguers on each team. "The realistic faces created by Microsoft texture mapper Rod Chang are one of the defining features of our model," Kovarik boasts.

GET READY FOR
THE MOST BRUTAL
HARD-HITTING
BLOODY ACTION
YOU'VE EVER SEEN.

AND THAT'S JUST
THE REVIEWS.

Introducing GamePower.
The game site with killer reviews and everything else you need to win.



Stay ahead of the game.

www.gamepower.com

CMP



Players can bat as their favorite slugger in their hometown ballpark.

"It was challenging because people know what these ballplayers look like," admits Chang. "It's important that they recognize the players when they see them, not just by seeing their names on the screen." Chang starts with photos of the players, arranging facial features to create textures to be applied to the model. Since the geometry is identical for each ballplayer, distinguishing features that might be structural — high cheek bones, a strong jaw line — can only be achieved by attention to the face texture.

Chang is also in charge of creating texture maps for player uniforms. He started by bringing in a real uniform to observe exactly how it fits and creases. Noticing that pinstriped uniforms weren't simply composed of vertical stripes — that pinstripes actually met in certain seams — he carefully arranged for the uniform textures to reflect this. More than just capturing photorealistic details, he strives for drama and verisimilitude by simulating how uniforms would look in the bright sunlight of summer afternoon, adding appropriate highlights and shadows to textures.

Of creating all the necessary jersey details, Chang says, "We had to do it because it made the game look more realistic," but admits it was the most tedious part of the job. Individual texture maps had to be created to represent names and numbers for each of three jerseys worn by 25 different ballplayers on all 28 big league teams. Modeler Randy Reeves fashioned a separate torso for ballplayers with single-digit jersey numbers, as those textures required different placement.

E-Motion

Of course, even with painstaking and skillful modeling and texture-mapping, a 1,000 polygon human figure isn't going to impress anybody

till it's set convincingly in motion. BASEBALL 3D will make use of extensive motion-capture data to bring its ballplayers to life with around 500 motion routines. Though I tend to give artists most of the credit for the visual success of a game, I've got to give a nod to techs John Pella, Microsoft's motion capture technical director, and programmer Stuart Rosen at Wizbang!, who are taking the use of motion-capture animation to the next level.

Two players from the Triple-A Tacoma Rainiers — Mike Knab and Sal Rosado — were brought in to Microsoft's in-house motion capture facilities to "go through the motions." Program manager and designer Rich Choi points out that particular attention was paid to miscellaneous motions that really add character to the game. Though the axe of time may fall on some of this plethora of kinetic personality, Choi is looking at including 15 batting stances (so you'll definitely see the difference between Jay Buhner's low hand-open stance and Edgar Martinez's high, looping stance), 36 swing motions (so a swing at an inside pitch looks different than a swing at an outside pitch, and so on), 4 pitching motions, 6 home-run trots, 4 running speeds, and numerous catch-and-throw motions such as thrown-from-knees (Caminiti-style), glove-toss, spitting on the ump (well, maybe not), and so on.

Using proprietary software, John Pella's motion-capture lab at Microsoft processes, optimizes, blends, mirrors, and loops all this data. This makes it possible for programmers to easily blend or interpolate among a library of captured motions. Softimage is used for editing and additional keyframing, but hand-editing is kept to a minimum by implementation of programmatic solutions wherever possible. "Writing a filter that performs some editing task on a huge batch of motions not only saves an enormous amount of time," Pella observes, "but it can be reused," reducing the cost and complexity of incorporating motion capture on future projects.

Motion-capture animations are integrated with player behaviors under Wizbang!'s ADLIB engine. Wizbang! programmer Stuart Rosen explains that, in object-oriented fashion, behaviors as well as geometry and texture-mapping characteristics are "owned" by the individual ballplayers. Rosen has developed a bipedal simulator as an extension of

the ADLIB engine, which uses motion-capture animation data as a source for producing true bipedal motion for the ballplayer models. The result: Players actually walk and run rather than slide across the field surface, producing a much more realistic experience.

Get Real

It's the new breed of graphics acceleration boards that act as the opener for the whole "realism" can of worms. Hardware acceleration allows higher polygon throughput and higher texture memory, which, at least in terms of system performance, enable artists to invest environments and characters with greater levels of detail. Microsoft texture artist Rod Chang puts it in perspective: "Working with a game that didn't require hardware-acceleration, the main objective of the artist was to make it as real as possible with the limited texture budget. For Baseball 3D, the challenge is in finding enough time to create all the detail that could not have existed in previous games."

But as designer Rich Choi avers, hardware-accelerated 3D is the future, and before long all new 3D titles will require 3D acceleration. "It also makes our job much more interesting and challenging."

If you think opting for a real-world setting and characters is taking the easy way out, the response from the BASEBALL 3D team would probably be "Get real!"

This is Dave Sieks' last regularly scheduled installment of *The Artist's View*, a column he has authored since 1994. He leaves the column to devote more time to outside projects — some of which have absolutely nothing to do with computer games, if you can believe such a thing — but will continue as a contributing editor to *Game Developer*. You can contact him via e-mail at dave@ward1.com.



Force-feedback joystick control will let players feel the difference between running on infield grass or baseline dirt.


I-FORCE™

The Force Feedback Standard

Version 2.0
Now with "DYNAMICS"



CRASH



BOING

Embrace the resistance...

I-FORCE is the latest gaming technology to take the computer industry by storm. When added to a joystick, steering wheel, or other gaming peripheral, **I-FORCE** enables users to FEEL jolts, blasts, liquids, springs, textures, surfaces, and countless other realistic physical sensations. These are not simple bumps or vibrations - **I-FORCE** provides complex physical sensations that replicate gaming reality. **I-FORCE** is currently targeted at PC, Arcade, and Console platforms.

I-FORCE has emerged as the standard technology for adding FEEL to gaming hardware. Developed by Immersion Corporation, **I-FORCE** is a standard architecture provided under license to makers of gaming peripherals. **I-FORCE** has already been endorsed by the following premier hardware makers:

CH Products

Interactive I/O

Nuby Manufacturing

Logitech

Thrustmaster

SC&T International

Advanced Gravis

InterAct Accessories

ACT Labs

You can expect many **I-FORCE** hardware products on store shelves by Christmas from the above manufacturers.

To find out more about **I-FORCE** or to see an updated list of hardware and software products supporting **I-FORCE**, please visit our web site WWW.FORCE-FEEDBACK.COM



BOOM



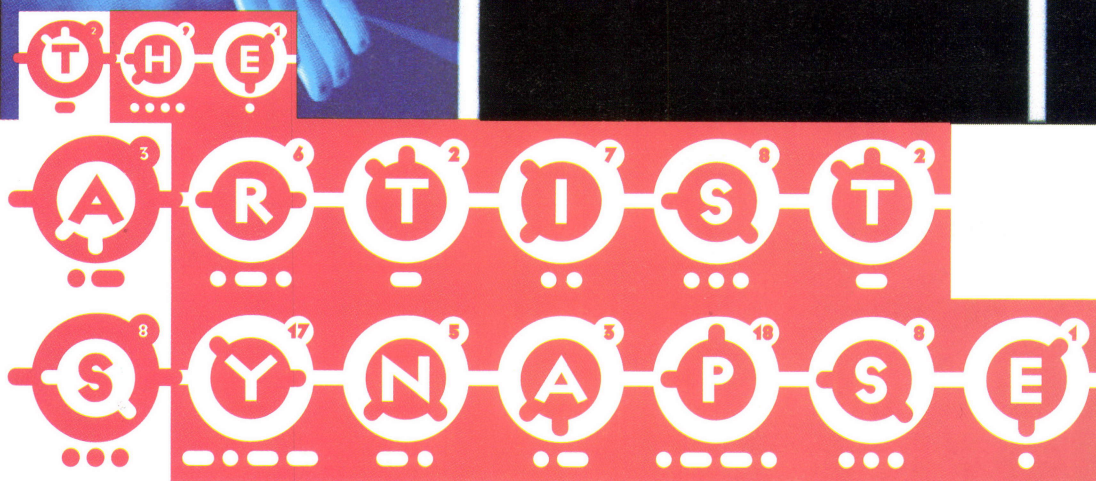
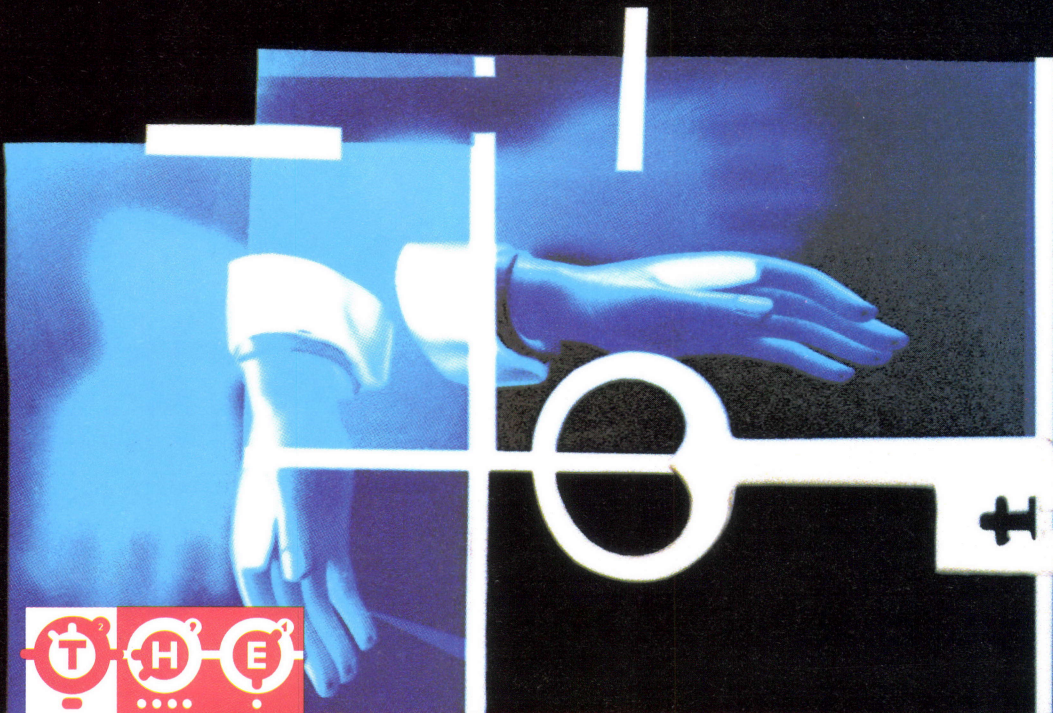
Immersion
Corporation

Immersion Corporation

2158 Paragon Drive, San Jose, CA 95131 USA


Tel (408) 467-1900 Fax (408) 467-1901

<http://www.immerse.com> info@immerse.com



TEAM BUILDING

BOY JOSEPH WHITE



Things are pretty rosy for us game developers — a fairly healthy industry, super-casual work environments, and creative, exciting new technologies to explore. But it's not all fun and profit: We've got our problems,

and there's plenty of room for improvement. Aborted projects are a major source of burnout, generally cutting into the fun and glory that game development should be.

One of the hallmarks of failed projects is communication breakdown. Why?

Because good communication is damned hard, and it's one of those mushy personal skills to boot. It's much easier to study tools than to learn to ask questions such as "how many textures can I use?"

This article teaches both skills to artists. Once we identify the artist's coworkers, we'll examine the psychology of the artist/programmer relationship, pitfalls, handholds, and those inevitable nuggets of advice.

A word of warning: I've sketched some big stereotypes in this article. They're not intended to be rude or to offend — I wrote them to help identify common interaction problems. So grit your teeth and hold your fire if you're insulted by the idea of stereotyping, but do speak up if you disagree with my basic generalizations. I write from

a wide and varied experience, but it's only one person's experience — more is always better.

Example Artist Interaction: Q&A for New RT3D Art Assignment

Before starting a new assignment, the artist needs to know specific information about the project. Without proper briefing, the art isn't going to match the needs of the

game. This major communication event provides a rich stew of interactions to observe. Since art assignments are very project-dependent, no complete, general-purpose checklist for eliciting the proper information has yet been developed. Nonetheless, I've created a hypothetical conversation so we have realistic information.

In my scenario, the experienced contract artist has just been hired to create artwork for a real-time 3D

game already in development. The artist is now getting acquainted with the project and its team members. As the artist asks questions of various team members (in the left-hand columns), I analyze the answers from both a psychological ("psych") and a technical ("tech") standpoint, which you can see in the right-hand columns. This will help you understand the context of and motivations behind the answers.

Artist: What's the basic idea of the game?

Producer: We're building a simple, real-time 3D comic strip about a cat that is hunting a bird, as the bird sits in its cage. The style is simple with somewhat realistic rendering. It's a cartoon, but we don't want any drastic squash and stretch effects.

Psychological Interpretation:

Contractors are often brought into a project without being told any more than necessary. The artist needs to know some basic context to do his or her job.

doesn't want to commit to a list, in case something else comes up later.

However, the artist should continue to find out as many specifics as possible: What kind of animations are needed?

What's in the room? These details will definitely affect the scheduling. Also, this is a good time for the artist to vaguely describe the planned style (quick examples) to make certain that the designer likes it. This kind of early communication can prevent disagreements about style.

Artist: What artwork do you need, exactly?

Designer: A bird, a cat, the birdcage, and the room, plus some animations.

Technical Interpretation:

The purpose of this question is to clearly define the assignment. If the artist had more design freedom, he or she would make a detailed list of objects to build.

Psych: This answer is quite brief, possibly because the designer hasn't thought out the scene in much detail. Also, this designer

Artist: What's the face/polygon (or vert) budget?

Programmer: For the cat, the budget is 250 polygons.

Tech: The basic geometry budget must be known up front before any work begins. Like most budgets, the polygon count is usually approximate, since its purpose is to guarantee a certain frame rate for the final game. A cat can be created using 250 polygons, but it won't be very detailed, so let's hope the engine has good texture-mapping capabilities.

Psych: This is another question that can make the artist feel trapped, because it's where the rubber meets the road. Often, programmers don't know the polygon budget or make up untested numbers. This uncertainty could be the sign of a basic design problem (which means it could be a sensitive issue). Experienced artists will gently explore the "why" of this budget and see if it matches their intuition.

EVOLVE OR DIE!

EQUIP YOURSELF
FOR THE REALTIME 3D
REVOLUTION



GameGen II
Realtime 3D Authoring Software

Need the Right Tools for Realtime 3D?

GameGen II was specifically conceived and designed for building fantastic realtime 3D game worlds for your target hardware. GameGen II is packed with the kind of power tools you need to meet tough time-to-market schedules. GameGen II puts the right tools for realtime in the hands of your artists. And, MultiGen's OpenFlight™ data format will help your programmers put peak realtime 3D performance in your game.

Need More Proof? Check out these hit titles built with MultiGen tools:

On PC - Twisted Metal 2™, Jet Moto™, Death Drone™, Cyberia™ 2
On Sony® Playstation™ - Warhawk, Twisted Metal™
On Nintendo 64® - Super Mario 64™, Super Mario Kart 64™, Wave Racer 64™
In the Arcade - San Francisco RUSH™
In LBE's - Disney's Aladdin™ VR Adventure, Magic Edge™



MultiGen and Industry
Experts present the Total Solution
Seminar Series on Realtime 3D for
entertainment. For more information,
check www.multigen.com
or call (408) 261-4100.

MultiGen Inc.®

THE REALTIME 3D COMPANY

Phone: 408 261 4100
email: sales@multigen.com
www.multigen.com

NOW HIRING TALENTED PEOPLE.
Email your resumé to: jobs@multigen.com

OpenGVS, just add water*

*... well almost... to get realtime, graphics hardware accelerated 3D games, you may need to add a dash of programming, a pinch of 3D models, a little audio, some networking and textures to taste. But in any case, you can save months of development time, without sacrificing framerate performance, by using OpenGVS as the realtime 3D scene manager for your upcoming graphics hardware accelerated game.

Mango Grits did – and it's enabled them to develop their award-winning 3D coin-op, LBE and consumer PC game in record time. Why? Because OpenGVS provides all of the realtime scene management features, including all of the hard stuff like object level of detail, dynamic texture paging, object culling, 3D-OOP, collision detection, LOD and texture management, visibility culling and damn near everything else you need to take advantage of today's hot new PC graphics accelerators.

Stop by the Quantum3D booth at SIGGRAPH '97 (booth no. 2057) and register to win a Quantum3D Obsidian 100-4440 and OpenGVS Evaluation Kit (\$2,500 Value). While you're there take a ride on the Mango Grits/Jessler LBE motion seat or visit their web site at www.mangogrits.com and see what OpenGVS can cook up for you.

For more information, check out <http://www.gemtech.com> or call us at (714) 598-0961; FAX (714) 598-0966. Also you can send email to info@gemtech.com.

3Dfx Glide Support

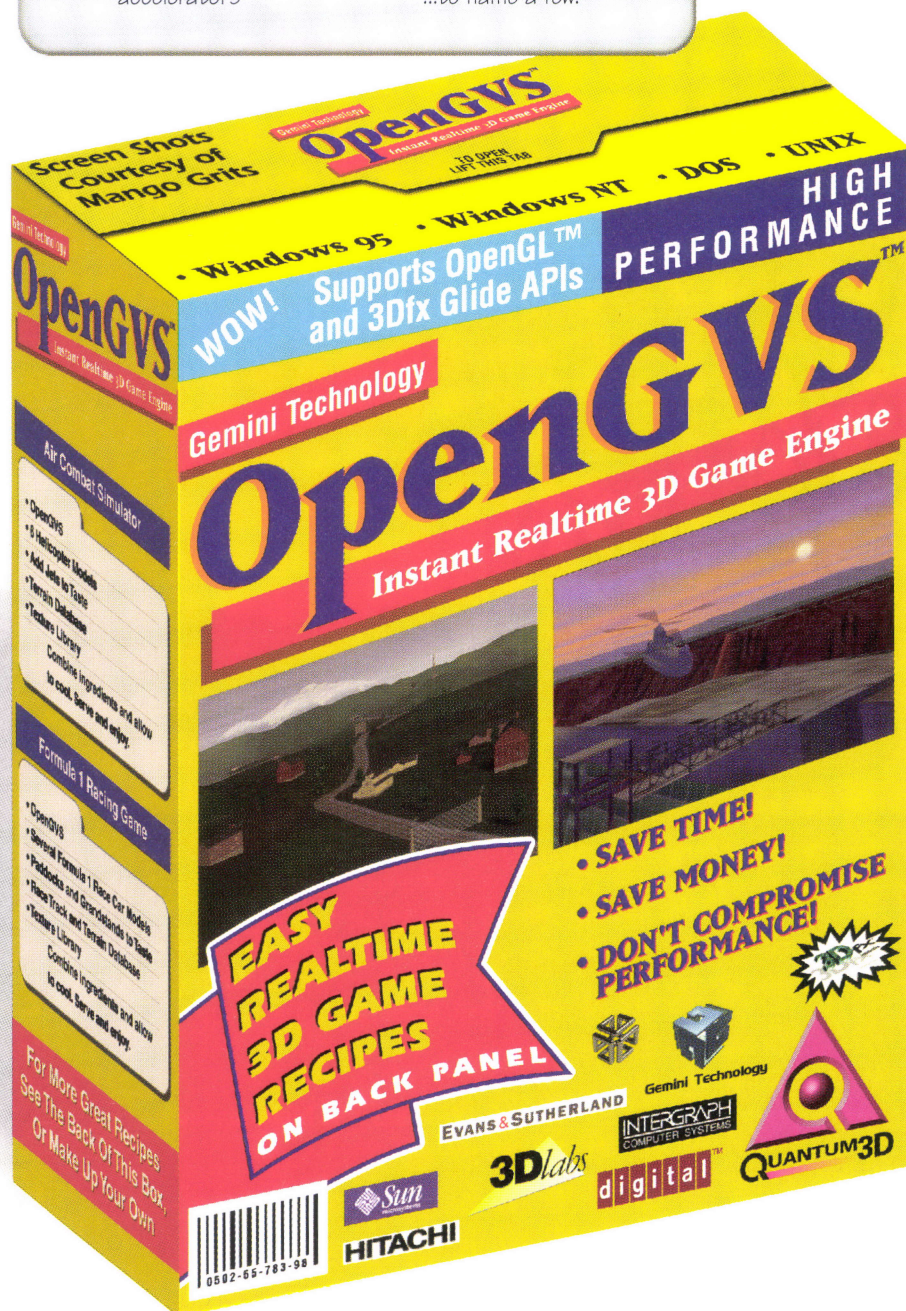
OpenGVS supports 3Dfx Interactive's Glide API -- so your game runs NATIVE on consumer and professional level 3D graphics accelerators based on Voodoo Graphics™ and Voodoo Rush™ including...

- Quantum3D's Obsidian™ and Ventana™
- Diamond's Monster 3D
- Orchid's Righteous 3D
- Deltron's Flash 3D
- Hercules' Stingray128/3D

OpenGL™ Support

OpenGVS supports OpenGL™ -- so your game also runs on workstations and high-end 3D accelerators that your developers and artists use for content creation, including...

- Silicon Graphics IRIS
- DEC workstations with PowerStorm Graphics
- Sun SPARCstations
- 3DLabs GLint and Permedia based OpenGL accelerators
- Real3D Pro 1000 and 100
- Intergraph Intense3D, Z and V
- E&S/Mitsubishi 3DPro based 3D accelerators
- Hitachi Spherix
- ...to name a few.



Artist: What kind of polygons are supported?

Programmer: Convex planar polys are supported.

Tech: The usual answer to this simple question is either "Only tris [three-sided polygons] are supported," "Quads [four-sided polygons] and tris are supported," or "Convex planar polygons with any number of sides are supported." When tris are used, geometry doesn't have to be planar and convex. This type of geometry is easier for the artist to work with, but requires more tris to build a model. Quads and n -sided polygons must be planar and convex, but fewer polygons are necessary to construct planar objects.

Artist: How are the polygons created from tris?

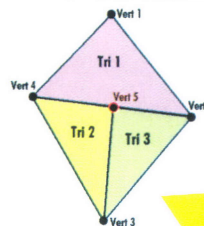
Programmer: If two tris share an invisible edge, they will be merged to a quad.

Tech: This is a pretty standard solution to the problem that some modeling tools have, most notably 3D Studio MAX and 3D Studio 4.0. The problem is that the tools don't directly support any polygons besides tris. Our hypothet-

ical programmer's solution isn't the only way to merge tris into polygons. For instance, sometimes a part of the art import procedure accepts a 3D data file, looks for coplanar tris that share edges, and replaces them with polygons.

Tech: This question applies if the import procedure has the ability to accept slightly noncoplanar quads as coplanar. This can save a lot of unnecessary editing, since it's common during normal construction for some faces to be slightly noncoplanar; it's difficult to make all faces perfectly coplanar.

FIGURE 1. T-sections.



Artist: What tolerance, if any, is used when deciding that a quad isn't planar?

Programmer: One degree or less out of plane is O.K.

Tech: The most difficult aspect of this question is agreeing on what a T-intersection is. As Figure 1 shows, T-intersections occur when one face touches another face, either in the center or at an edge, without sharing all vertices.

If the answer is "no," the artist needs to understand the consequences because sometimes T intersections are almost impossible to prevent.

Artist: Are T-intersections allowed?

Programmer: Yes.

on intersections. Be sure that the programmer agrees on the definition of "intersecting faces." What artists generally need to know is whether two faces can cross each other, forming a line (or a plane if they are coplanar) of intersection.

Artist: Are intersecting faces allowed?

Programmer: Yes.

Tech: This question may be asked instead of the sorting question that follows, but an understanding of the various sorting options is preferable to a simple yes or no answer

Degrees of Separation

Remember the movie *Six Degrees of Separation*? It was a heartwarming story about some poor guy who scammed his way into high society, with the shocking conclusion that poor people are human after all. Anyway, the title expresses an intense idea: Everyone is a friend of a friend of a friend of a friend of a friend of a friend of a friend (hence the term "six degrees").

Let's apply this idea to the artist hard at work on a game. The artist's first-degree contacts are the developers that the artist interacts with daily, as shown in Figure 2; second-degree contacts perform functions such as testing and sound production, as well as nondevelopment things such as legal, marketing, and so on.

- **The Art Director.** This person tells the artist what design/style of art to create and reviews the work for visual quality.
- **Artists.** Peers who interact as they work. Other artists who may be working off site report directly to the art director.
- **Tool Programmer.** This person tells the artist exactly how to import art into the game (and usually writes that code).
- **Game Designers.** The designer decides what art pieces are needed for

the specific game experience.

- **Producer.** Generally, the person in charge. The producer makes decisions about what the game will and won't consist of, in terms of its features and the overall look and feel. They also coordinate other development efforts, such as sound and testing, and act as a firewall for publishers, lawyers, press, and other nondevelopers involved in the project.
- **Lead Game Programmer.** This person decides what the game can do (and codes the functionality of the game). Depending on the team, the lead programmer may or may not directly interact with the artist.
- **Project Manager.** The project manager keeps track of the schedule, equipment, and costs. Sometimes, the producer takes on this role.

Commercial artists have been trained to impress their audience, not their peers.

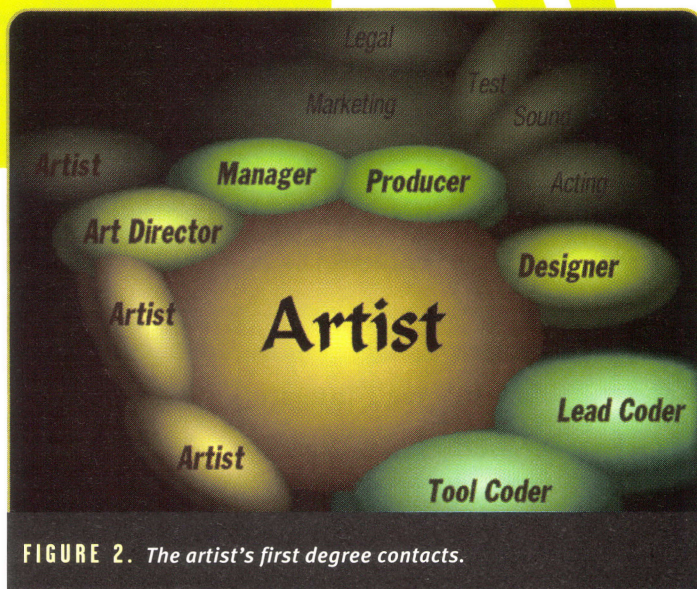
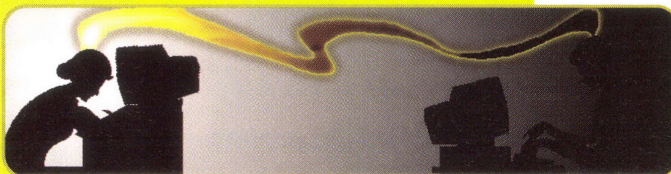


FIGURE 2. The artist's first degree contacts.

Artist: What kind of sorting is used? Per-pixel Z-buffer? Per-face BSP-tree sorting?

Programmer: Z-buffer sorting, with object culling.

Tech: The main purpose of this question is to determine whether intersecting faces are supported (Z-buffering allows intersecting faces, but most other sorting methods do not), but the answer can affect other areas as well.

Artist: Are there any special shapes that should not be built?

Programmer: No line-face, point-face, or 1,000:1 skinny faces.

Tech: This question reflects the fact that some graphics engines don't cope with strange faces, such as a "line-face," which only uses two vertices, or a "point-face," which only uses one vertex. Also, some engines don't handle long, thin faces well (such as a triangle that fits in a 1,000x1 rectangle). Asking this question should reveal all those special cases, if there are any.

Artist: What's the screen size for this application?

Programmer: 640x480

Tech: This very important question tells us what resolution the textures should be, and can also be used to determine the size of detail the on-screen objects will need. For example, if the answer was 320x200, then a smaller fur texture might be O.K., and perhaps the gaps between the teeth wouldn't be worth modeling, since they would always be far less than one pixel wide.

typical viewing distance will be, and what the field of view is.

Tech: With enough information, the optimal size for an object is a known quantity. In this example, multiply the screen width (640) by the cat's length onscreen (25%) to conclude that the cat's optimal size is about 160 pixels long. This number is useful when choosing texture sizes (128x128 for the foot is way too detailed) and geometry detail.

Artist: How will the models be seen?

What viewpoints does the player use during the course of the game?

Designer: The camera will orbit the cat, barely within the room. The cat will rarely take more than 25% of the screen width, and usually only 10% or so. We won't see the cat's underside very often — the view will be mainly from a six-foot-tall person's standpoint, much as you'd see a real cat in a living room.

Psych: This is probably the hardest question to which to get a straight answer. Of course, RT3D means the viewpoint usually isn't defined, and the designer can't know the viewing positions. The good news is that artists only need to know very general information: what parts will be scrutinized, and which are rarely seen. Ideally, they will know more: how and where the object will be in the scene, what the minimum, maximum, and

Artist: What kind of animation will this model be able to use?

Programmer: 3D morphing with support for bones.

Tech: The most common types of real-time animation include hierarchy, animated bitmaps, 3D morphing, or "none at all." This question will probably elicit a lot of detailed discussion on how to build the animation. Once the animation type has been settled, it's a good idea to build a simple test case and make sure it works.

Psych: In our example, we're getting a pretty sophisticated real-time animation system. That probably explains the low face budget — fancy animation methods usually hurt performance. It might be a good idea to verify this theory with the programmer.

Artist: How do I get animation data into the engine?

Programmer: 3DS morph animation data is directly supported, so simply save the morphing animation sequence with the file name of the action — STALK.3DS, POUNCE.3DS, and so on.

Tech: This critical question tells the artist exactly how to import animation data — it's usually much harder than importing the geometry and textures. For example, sometimes the artist must supply a string of 3D models as keyframes (STALK01.3DS, STALK02.3DS, and STALK03.3DS), or keyframes can be imported directly out of the 3D Studio animation data, as our answer indicates. The answer is usually followed a torrent of questions about how to handle animation data. For instance, what kinds of bone motions are possible — keyframes or every frame? Branching? Transitions? This issue requires a separate article to explain, but it's very important.

Artist: Will any moving objects always be moving?

Programmer: Yes.

Tech: For objects that have moving parts, such as spinning wheels on a car or propellers on an airplane, we need to know if the spinning object will ever be seen still, which would require a second model (blurred and still).

Tech: The usual answers are 8-bit (paletted), 16-bit (high-color) or 24-bit (true-color). This issue may be somewhat more confusing if the game is going to be used on multiple platforms.

Artist: How many colors is the target platform capable of showing?

Programmer: We're using 16-bit color.

The Lost World: Jurassic Park

Image courtesy of Industrial Light & Magic. ©Amblin/Universal. All rights reserved.

Virtua Fighter 3

©SEGA Enterprises, Ltd.



Cherry Baby
Images courtesy of Digital Domain

The Fifth Element

Image courtesy of Digital Domain.
©Columbia Pictures Industries. All rights reserved.

(800) 576-3846 x324

(818) 365-1359 x555

ReBoot™ & ©1997 Mainframe Entertainment, Inc.

The ReBoot™ name and images are used with the permission of Mainframe Entertainment, Inc.; and all characters are based on the animated television series "ReBoot™", an Alliance/Mainframe Entertainment, Inc. production.

All other trademarks belong to their respective owners and are hereby acknowledged.

SDK
seminar
registration

Digital Studio Technologies
synthesizing into a single
creation platform for both
Integrated, Digital Studio
offering new levels of creation
taking advantage of the V
Studio is easily extensible
broad support for plug-

Soft **02**

SOFTIMAG

All stories

SOFTIMAG
imageque.

97

LOS ANGELES



Product and Training Information

in North America: Worldwide:
(800) 576-3846 x324 (818) 365-1359 x555

Customer Support

in North America: Worldwide:
(800) 387-2559 (514) 845-2199

Web

www.softimage.com

Sales Offices:

Western US

Western Canada
T: (510) 803-2300
F: (510) 803-2301

Eastern US

Eastern Canada
T: (203) 830-6355
F: (203) 830-6359

Latin America

T: 55 11 5514 7286
F: 55 11 5514 7106

UK

T: 44 171 287 0708
F: 44 171 287 0701

France

T: 33 1 46 04 33 00
F: 33 1 46 04 39 61

Germany

T: 49 261 912 610
F: 49 261 912 6133

Italy

T: 39 2 70392524
F: 39 2 70392020

Asia

T: 65 336 6455
F: 65 336 3335

Japan

T: 81 3 5454 8068
F: 81 3 5454 8164

Credits

A: Image courtesy of A.D.A.M. Software
 B: "Obsidian" Animations ©1997 Segasoft
 C: "Hexen2" Image courtesy of Raven Software
 D: "When Jakob woke up, he was different" Claudius Brodmann
 E: "FIFA97" Electronic Arts, Inc.
 F: Image courtesy of SVC, London
 G: "The Fish" Sean Henry
 H: "Obsidian" Animations ©1997 Segasoft
 I: "Fly Hard" Gino Della Savia, Peter Harelabus,
 Nick Michaleski & Greg Tareta
 J: Codename: Tenka ©1996 Psychosis. Courtesy of Tenka team
 K: Image courtesy of cinaframe animation / David Gallagher
 L: Digital Animation and Compositing by Little Fluffy Clouds
 M: "Obsidian" Animations ©1997 Segasoft

SOFTIMAGE® VIRTUAL

A: Image courtesy of A.D.A.M. Software
B: "Obsidian" Animations ©1997 Segasoft
C: "Hexen2" Image courtesy of Raven Software
D: "When Jakob woke up, he was different" Claudius Brodmann
E: "FIFA97" Electronic Arts, Inc.
F: Image courtesy of SVC, London
G: "The Fish" Sean Henry
H: "Obsidian" Animations ©1997 Segasoft
I: "Fly Hard" Gino Della Savia, Peter Haralabous,
Nick Michaleski & Greg Tareta
J: Codename: Tenka ©1996 Psychosis, Courtesy of Tenka team
K: Image courtesy of cinaframe animation / David Gallagher
L: Digital Animation and Compositing by Little Fluffy Clouds
M: "Obsidian" Animations ©1997 Segasoft



SOFTIMAGE® VIRTUAL

performance and reliability. Softimage|3D. The company oversees the HCT production tests, distribution, data

SCAVENGER HUNT

FIND THE MAGIC

images on our web site and
we'll send you a cool prize*.

See our homepage for details.

Microsoft

© COPYRIGHT 1997 MICROSOFT CORPORATION. ALL RIGHTS RESERVED. PRINTED IN CANADA. SOFTIMAGE® IS A REGISTERED TRADEMARK OF SOFTIMAGE INC., A WHOLLY OWNED SUBSIDIARY OF MICROSOFT CORPORATION, IN THE UNITED STATES, CANADA AND /OR OTHER COUNTRIES. MICROSOFT®, WINDOWS®, AND WINDOWS NT® ARE REGISTERED TRADEMARKS OF MICROSOFT CORPORATION IN THE UNITED STATES AND / OR OTHER COUNTRIES. ALL OTHER TRADEMARKS BELONG TO THEIR RESPECTIVE OWNERS AND ARE HEREBY ACKNOWLEDGED.

*WHILE QUANTITIES LAST

The Lost World: Jurassic Park

Image courtesy of Industrial Light & Magic. ©Amblin/Universal. All rights reserved.



The Fifth Element

Image courtesy of Digital Domain. ©Columbia Pictures Industries. All rights reserved.



For more information about Softimage:
in North America: (800) 576-3846 x324
Worldwide: (818) 365-1359 x555
www.softimage.com



ReBoot™ &©1997 Mainframe Entertainment

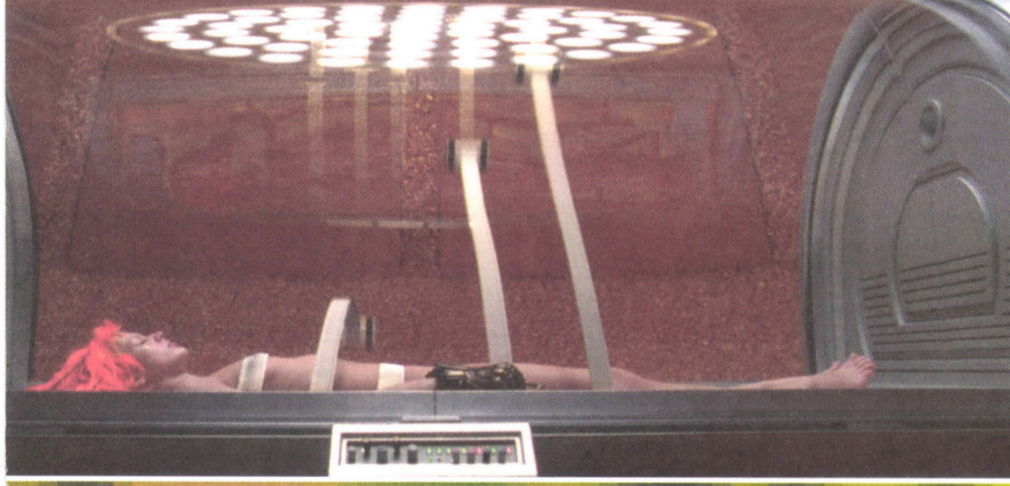
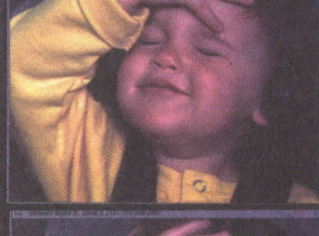
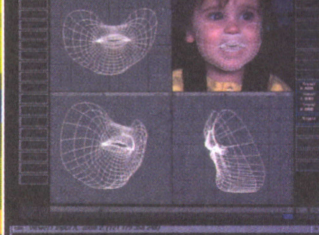
The ReBoot™ name and images are used with the permission of Mainframe Entertainment, Inc.; and all characters are based on the animated television series "ReBoot™", an Alliance/Mainframe Entertainment, Inc. production.

All other trademarks belong to their respective owners and are hereby acknowledged.

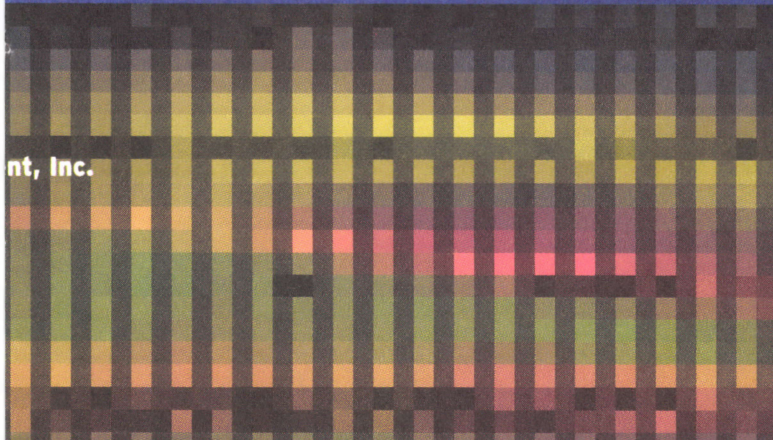
Virtua Fighter 3
©SEGA Enterprises, Ltd.



Chevy Baby
Images courtesy of Digital Domain



owered by passion



The best way to speed graphics programming just got better.

SciTech MGL 3.1

Professional Graphics Libraries for DOS and Windows 3.1/95/NT

The Ultimate Graphics Libraries

The SciTech MGL 32-bit graphics libraries give you direct access to any drawing surface so you can mix and match the SciTech MGL's graphics routines with any of your own drawing or rendering code to come up with a high performance application in the shortest time. SciTech MGL primitives work identically whether you are rendering directly into VRAM or into a system buffer. Your SciTech MGL code is fully portable between DOS and Windows 3.1/95/NT versions (other OSs planned).

New Since v.2.0

- Seamless DirectX support
- Return of ModeX support
- More low resolution graphics modes for fast software 3D and digital video
- Hardware scrolling/panning surfaces
- Sample MGL Smacker Video Player
- Watcom Win386 mode support
- DJGPP 2.0 support
- Borland Delphi Support

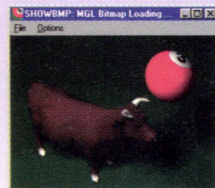
Includes SciTech MGL/Lite

- Only initialization/setup functions
- Perfect for custom rendering engines
- Smaller footprint, less code



*SciTech MGL
3.1 supports
transparent
sprites*

*SciTech
MGL 3.1 lets
you use
hardware
acceleration*



*SciTech MGL 3.1
does on-the-fly
color space
conversions*

FREE DEMO

Fully functional evaluation versions of SciTech MGL can be downloaded free of charge from our web site. Or, call our toll-free number and we will send you an evaluation CD-ROM. You pay only shipping and handling charges. Those charges are fully applicable toward purchase of SciTech MGL within 30 days.

Featuring:

- Highly optimized 32-bit assembler rasterization
- Color depths from 8 to 32-bits per pixel
- Resolutions from 320x200 to 1600x1200
- DOS or Windows 3.1/95/NT versions
- Full screen graphics under Windows using WinDirect (Win 3.1/95) or DirectX (Win95/NT)
- Supports all standard C/C++ compilers
- Automatic detection and utilization of VGA, VESA VBE 1.2/2.0, VESA accelerator (VBE/AF), WinG, CreateDibSection or DirectX

2D drawing capabilities: lines, rectangles, circles, ellipses, polygons, triangles, quads, vector fonts, bitmap fonts, bitmaps, transparent bitmaps, arbitrary regions and mouse cursor support.

3D drawing capabilities:

lines, triangles and quads with flat shading, gouraud shading, 16/32 bit zbuffering and 8 bit RGB dithering.

ONLY
\$299⁹⁵
DOS or
Windows

For a free demo,
to order, or to locate a reseller call

(800)486-4823 (EXT 053)

www.scitechsoft.com



505 Wall Street, Chico, CA 95928 • Phone (916)894-8400 • FAX (916)894-9069 • Email: info@scitechsoft.com

© 1997 SciTech Software, Inc. All Rights Reserved. SciTech MGL and WinDirect are trademarks of SciTech Software, Inc. Quake is a registered trademark of id Software. All other trademarks are the property of their respective owners.

Artist: Are there any palette issues of which I should be aware?

Programmer: Nope.

Psych: The terse response here is because the previous answer pretty much guaranteed that this question wasn't relevant — 16-bit color almost never has palettes. The programmer probably thinks less of the artist now; a better way to ask this might have been, "I know 16-bit usually means no palette; are you doing anything special about palettes?"

Tech: This question always applies when the game uses 8-bit color. Again, this issue is too complex to explain fully here, but it's a very key point and there aren't many standard ways of handling palettes. The artist needs to know as much as possible about how the game's palettes work — specifically, how many (if any) of the colors can the artist dictate values for, how the fade tables work (if they exist), what happens when the lighting changes the colors, how to swap palettes, and so on. Also, the concept of palettes could be applied to higher color depths; even if we're in 16- or 24-bit, we may have to deal with palettes.

Artist: Are there any "special" colors? Transparent colors, for instance?

Programmer: Nope.

Tech: This question refers to colors that are handled by some special method. For example, some graphics engines consider any pixel that is pure black (RGB = 0,0,0) to be transparent, if the material has any file name under the "opacity map" setting. Palette-based systems often use certain palette entries for flashing colors, color-cycling, and numerous other effects.

Psych: Another terse answer — this may be because the programmer's irritation (read: loss of respect for the artist), which resulted from the last question. For example, the programmer might assume the artist is using "color" to mean a palette entry, not RGB values. If so, "nope" would mean, "No, dammit, we're *not* using palettes," rather than, "No, we're not doing any special-effect colors." Again, the solution is to demonstrate knowledge of the issue in the wording of the question.

Artist: What kind of lighting will be used in the game?

Programmer: Static point-sources. Actually, "static point-source lights with ambient" is more accurate. Lights must be defined in the LIGHTS.DAT file using the format lightname,XYZ,Radius,RGB,falloff. Multipass lighting is also supported.

Tech: As you can see, the information relevant to this question includes the lighting type (point-source, directional, ambient), how to set these lights, and whether the lights can change during run time (dynamic) or not (static).

Artist: How should dynamic lights be animated?

Programmer: Dynamic lights aren't supported.

Tech: If dynamic lights are supported, the artist will need to know how to define their animation. Hopefully, the artist can simply animate in the modeling software and export that data — but it doesn't always work that way.

Psych: Oops! This is another question that seems to demonstrate artist ignorance. The previous answer said that the lights are "static," meaning not animated. More loss of respect for artist's technical knowledge... and perhaps rightfully so. It would seem that our artist really doesn't know what static means. The artist probably should have asked the programmer to define "static" when the term was first brought up, rather than stumble ahead as if it were all understood.

Artist: How should I create multipass lights?

Programmer: Create separate geometry that is coincident with, but simpler than, existing geometry. Map it with the lightmaps and name it MP-OBJNAME. At run time, this model's rendering will be multiplied with the basic model.

Tech: There are many ways to handle multipass lighting input, and no real standards except QUAKE, which only really works for QUAKE or QUAKE clones. The programmer's answer is a simple example; real projects are usually different.

Psych: By bringing this issue up, the artist is wisely exploring a part of the programmer's earlier answer that wasn't fully explained. The principles of multipass lighting aren't usually part of normal 3D modeling software, so this issue could probably use some more discussion, and even a simple test of the procedure.

Artist: What kind (if any) of texture mapping is supported?

Programmer: Perspective-corrected, no run-time filtering.

Tech: It's a rare game that doesn't support texturing. There are several kinds of texture mapping possible, each with different tradeoffs between rendering speed and distortion.

Artist: How much texture-map memory is budgeted?

Programmer: You have 2MB of texture memory

Tech: Obviously, the texture-memory limit affects how large and how many textures we can use. This is a pretty typical limit for 1997 graphics accelerator cards — usually, developers are limited to the texture memory on the card, which is often 2-4MB.

Psych: This question is a really important one, and the artist should make a strong effort to get as accurate a number as possible. If the producer is technical enough to answer meaningfully, the artist should probably ask the producer first. If it can be done casually, it would be worth exploring the memory situation with

the programmer a bit — asking what is in memory, if the Z-buffer uses VRAM, and other topics out of the scope of this article.

Artist: What size and shape can the texture maps' dimensions be?

Programmer: They've got to be 16x16, 32x32, 64x64, 128x128, 256x256, or 512x512.

Tech: These numbers are pretty restrictive — artists would prefer nonsquare textures (such as 64x16) and nonpower-of-two sizes (such as 234x11), because it allows the artist to be more efficient in using the texture memory.

Artist: What kind of shading is supported?

Programmer: Smooth shading.

Tech:

Almost every graphics engine is capable of flat shading from a fixed directional light source. Most can also handle Gouraud shading (a.k.a. smooth shading) and some can handle specular highlights (shiny spots). A few do Phong shading as well.

for flat-shaded (as compared to smooth-shaded) objects.

Psych: If the artist gets an curt, "Of course it all works," he or she needs to proceed with caution. The artist in this situation should create a simple test case and prove it before doing a lot of work that depends on a combination of these features. the prototype will also be a valuable debugging aid to the programmer, though he or she may not be thankful initially.

Artist: What are the allowable combinations of shading, lighting, and texturing?

Programmer: All can be combined.

Tech: This important question is often forgotten amid the confusion of technical answers, but it's important to ask. For example, some graphics engines won't smooth-shade a textured face, and ambient lighting may behave differently

Artist: How are Gouraud shading borders determined?

Programmer: Any faces that share edges are smooth-shaded together.

Tech: Most smooth-shading graphics engines assume that any faces with shared vertices should be smooth-shaded together.

The Psychology of Artists and Programmers

I've found that the programmer/artist connection is difficult for many artists. I've often heard artists complaining about trying to work with programmers, saying such things as, "They have no idea that they just ruined the whole look of the game with that limit!"

I think there is often a disconnect at two levels: One is an important difference in goals for artists and programmers, and another is the garden-variety difficulty in communicating with which all team members must cope. Let's explore the first disconnect in detail. Artists generally encounter two common problems when working with programmers (here come the stereotypes!).

FEATURE CREEP. "Feature creep" is when a product continually gains new features that weren't in the original design. This happens a lot with games, and it's often driven by programmers' improvements.

Why? Game programmers are very competitive with each other, and the score is often measured in features, not in depth of game experience. For example, I know high-profile coders that don't respect MYST because it isn't technically innovative. Lots of artists that I know think MYST is a successful game that has a powerful impact on its users. This extreme focus on the technical abilities of the game leads to ever-greater features, sometimes even at the expense of the overall game experience. By comparison, artists are more focused on the overall effect of the game: Commercial artists have been trained to impress their audience,

not their peers.

"FIELD OF VIEW." Programmers can lose the forest for the trees very easily. It's an occupational hazard — they have a death grip on microcosmic detail, and getting that detail to function is one of their primary tasks. This extremely heads-down approach means that it's hard to keep the overall importance of their current task in mind. For example, a programmer accidentally makes a renderer that generates cool lighting streaks when the player rolls. In the code, this looks like a mistake, and so the programmer fixes it... and kills a potentially valuable addition to the game.

Deciding what features to develop is really a technical design issue, not a coding implementation problem. Nonetheless, an industry tradition has arisen around this issue. In the early days, programmers designed and built

SERIOUS BUSINESS FOR SERIOUS GAMERS

NUVISION 3-D SPEX.™

LIKE STEREO FOR THE EYES.

3-D
SPEX

Stereoscopic 3D is the biggest thing in PC games since trilinear-filtered perspective-corrected texture mapping. It's the most effective way to give your games the wicked realism serious gamers crave. It's easy to implement and ready for your DirectX and DOS games today.

3-D SPEX stereoscopic eyewear brings your customers the same high-performance, lightweight, stereoscopic viewing technology used for over ten years in military and industrial applications. Crisp, three-dimensional depth cues literally wrap gamers into the action. If your game is great, it'll be killer with 3-D SPEX.

Open your eyes to the real thing in 3D: experience the best kick-butt stereoscopic 3D you've ever seen. Contact us at www.nuvision3d.com or 1-800-920-9327 and ask for your Windows® 95 or DOS SDK.

NuVision
TECHNOLOGIES, INC.



the whole game and subcontracted out other parts (including the art) at the end — not an ideal arrangement for the artist. Today, programmers are part of a development team, but they often decide the game's features and capabilities, sometimes without the benefit of an artist's (or level designer's, or sound producer's, or a host of other developers') input.

THINKING VISUALLY. Programmers don't think in the same visual terms as artists. As an artist, this point is driven home when I watch a programmer work. Standing over a programmer's shoulder, watching endless streams of text scroll past, I can see that the visually plain text formatting is not a significant barrier in their understanding. Of course, artists understand complexity — I've seen programmers shake their head as they watch an artist thoughtfully rotate a nightmare tangle of wireframe lines and points, then suddenly pounce on some apparently-problematic intersection. Both types are skilled at very complex, subtle problem-solving — the difference I'm emphasizing is the appearance. Artists understand visual problems, even if they're quite symbolic in nature, and programmers think in procedural flows that aren't necessarily visual at all.

HOW ARE GAME PROGRAMMERS SIMILAR TO ARTISTS? Programmers and artists are in a similar role: They're both major players on the frontline of development, caught between the realities of time, features, and managers' expectations.

They produce the actual product, and that makes them peers in the end. They have similar issues with milestones and deliverables, and also have similar knowledge and attitudes about the rest of the game production cycle (marketing, distribution, customer satisfaction, and so on).

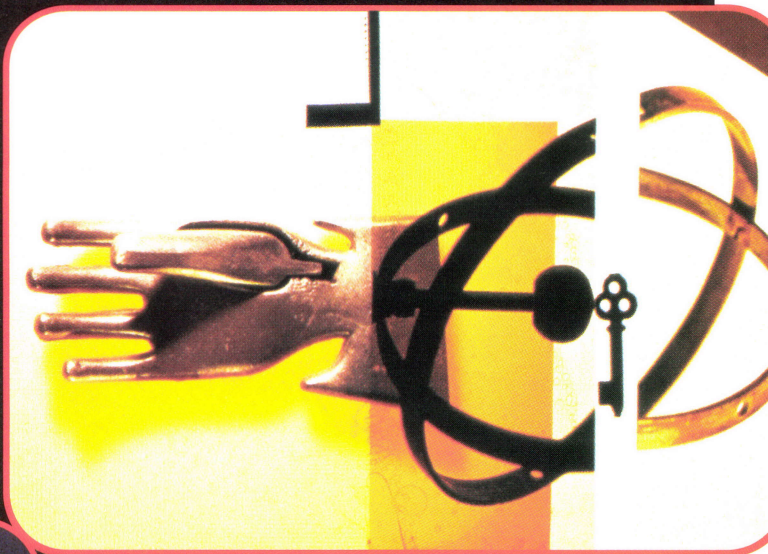
Both coders and artists are perfectionists, with their names in the credits. They usually believe in their own abilities, have a strong desire to produce a top-notch game, and are willing to work hard to do it and fight for what they think is important. For example, in real-time games, frame rate is a very important issue to both artists and programmers. They'll both howl when their game runs at a miserable ten frames per second, and they'll both have good insights and often volunteer to improve the situation. This commitment is a common bonding point for artists and coders — they develop respect for each other's dedication (without the danger of feeling competitive), sometimes expressed as a shared scorn for "slacker" nine-to-five workers.

means most people don't have this ability. Still, most artists that I know take this skill for granted, and find it frustrating when other people (say, programmers) ignore visual beauty. This lack of understanding is the root of many communication problems, and yet both programmers and artists share it.

Suggestions to Artists

I'm sure you think that's all very interesting, but what can artists do to improve the situation?

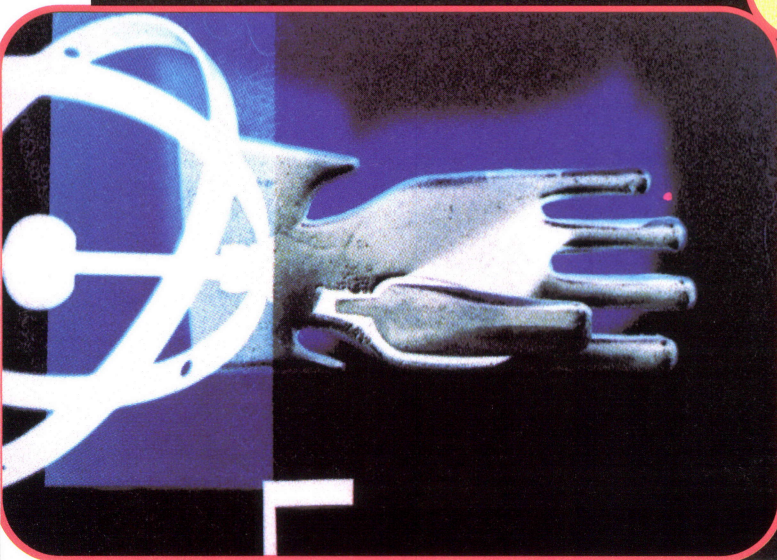
Get coders to understand your issues. For example, if you need more texture memory, don't just beg for more. Show them dramatic (but not faked) "before" and "after" textures so they can see the benefit of an improved texture and weigh it against



Like many specialists, artists and programmers can fail to understand what specialized knowledge they have. For example, professional artists have unusual training and talent that lets them see and express visual beauty. "Unusual"

the cost of the memory usage. If you can show a dramatic improvement, they'll be trying to figure tricky ways to get that cool art into the game. Be aware of the level of understanding that programmers have for varying parts of your work; don't explain concepts they already understand.

Be assertive about artistic judgment issues — remember, you're the expert at that. If a proposed feature doesn't add much to the environment, say so and recommend that it not be implemented (expressing this dissenting opinion without causing strife is tricky indeed!). In a similar vein, be respect-



ful of your colleagues' fields of expertise, unless you have very solid reasons to doubt them.

Keep programmers in your loop. Consult with them, in their terms, especially about issues that affect performance. For example, if you're trying to create an army of 100 action figures, but your face count is too low, show a programmer a sketch of the scene you want and run through a frame-rate calculation with them ("O.K., so 100 figures means 8,000 faces. The performance of 1,000 faces per frame means..."). This provides a reality check (and a solid justification if anyone questions your decision), and it also gives the programmer a chance to suggest an innovative solution.

Stay in your programmers' loop. Excellent artists know the limits of their medium no matter what the art form. Also, tech-savvy artists can see improvements that others can't. Becoming technically savvy isn't diffi-

cult if you can find a friendly, communicative programmer.

Don't waste excessive hours addressing unimportant or unrelated issues. If you have a hard time distinguishing, don't suppress your queries, but do confine them to lunch-time or off-hour conversations.

Features vs. Beauty

Here's my theory: In general, programmers must balance features, performance, and time — they want the most capability, at the best frame rate, within their deadlines. On the other hand, RT3D artists are trying to balance artistic quality, performance, and time — they're trying to get the best looking art at the target frame rate, within their deadlines. This difference is the root of many programmer/artist woes.

In the worst case, coders push for features without caring if the game is

any better, while the artists ignore these new features as they build detailed artwork in their own way. They don't talk much during development, then try to integrate their work at the end, which climaxes in a grand fight when the game sucks.

Of course, it's not normally so bleak. Programmers implement features that allow the game to show better looking art without compromising frame rate, and artists take advantage of these features to produceable. ■

Josh White (josh@vectorg.com) has been building real-time 3D models for games since 1990. He runs Vector Graphics (a company devoted to creating RT3D art), cofounded the CGA (a community of computer game artists — www.vectorg.com/cga), wrote Designing 3D Graphics, the first book on real-time 3D modeling, gives lectures at CGDC, and writes about computer artists... but he admits that all this really kills time between soccer games.



- ◆ Create 3-D action, adventure, role playing games and publish them **royalty free!**
- ◆ Contains World Editor and **ACKNEX** 3-D engine - no programming skills required
- ◆ Free 3-D template game with **150 free textures**, sounds, weapons, enemies included
- ◆ True **vector-based** topology (no tiles!) for creating slopes, bridges, dungeons, multi-floor buildings; insert **your own** objects, actors, enemies, traps, and weapons
- ◆ Lightsources, light direction, ambient shading, transparent & translucent textures
- ◆ Objects can be manipulated with the mouse (e.g. to collect them in an **inventory**)
- ◆ Player can walk, drive, fly, climb, swim, dive... with realistic movement physics
- ◆ Self-defined titles, menus, overlays, panels and scrolling text on up to 16 layers
- ◆ **320x400** smooth scrolling VGA resolution, **8-channel stereo** sound, midi support
- ◆ Imports PCX, LBM, WAV, MID and IBK files
- ◆ 200+ page English manual with game tutorial

3D GameStudio lite \$99
3D GameStudio commercial (SVGA, 2-player-mode) \$199

CONITEC DATENSYSYSTEME
 D-64807 Dieburg/Germany • Dieselstr. 11c • Tel +49 6071-9252-0 • conitec@aol.com

Prices **+\$20** overseas shipping
 More information, demos, ordering:
 → <http://www.conitec.com>

WE'LL BLOW YOU AWAY

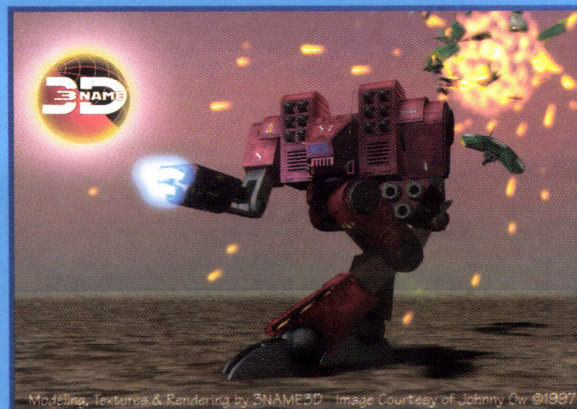
For all of your
 Custom CGI
 Modeling & Game
 Development needs



3D Modeling
 Texture Creation
 & Mapping
 Sculpting & Digitizing

<http://www.3name3d.com>
1-800-993-4621

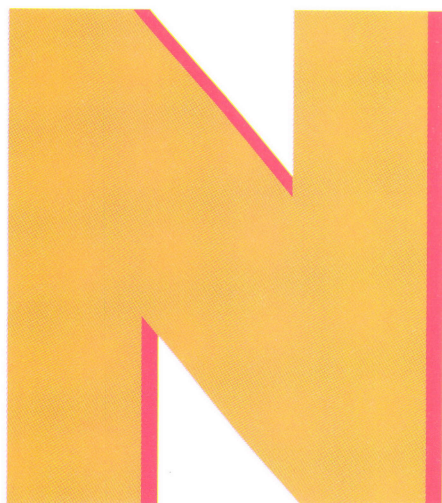
fax: 310-314-2181 email: info@ywd.com



3D Modeling & Textures to fill your Game World Needs

Come See us at SIGGRAPH '97! booth #1517

3NAME3D... THE ARCHITECTS OF CYBERSPACE.



etworked computer games present new data distribution challenges to the developer. With stand-alone games, all game information is available in one place, on one computer, all the time. When designing networked games, however, the developer has to consider what information the

other players on the network need, when

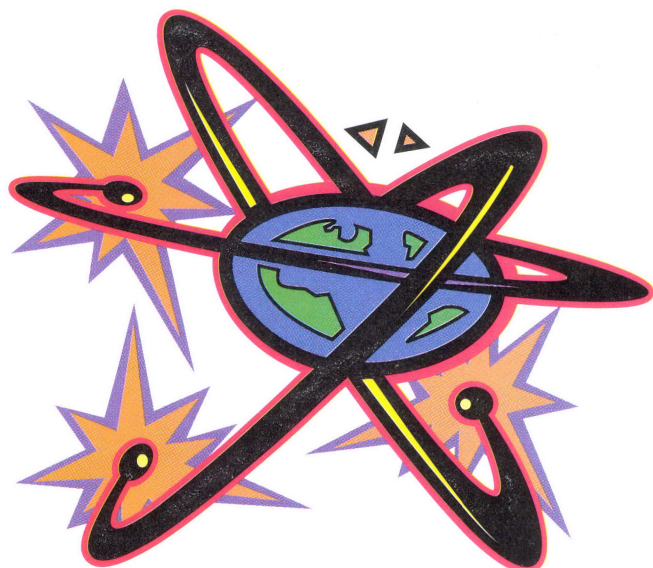
they need it, and how to deliver it to them. The easy way out is to broadcast everything you can think of to everyone on the network as often as possible. This keeps everyone synchronized, but quickly consumes all available bandwidth — not a good idea for groups of players dialed in to the Internet. The other extreme is to carefully consider each packet of data and to send it only to other players who really need it. This approach makes efficient use of bandwidth, but deciding who needs data and who doesn't chews up so many of the sender's CPU cycles that no processing power is left to play the game.

There is a middle ground, however, and it's called "grouping." Grouping allows networked games to route essential data among the players while making efficient use of both network and CPU resources. In some cases, underlying network service providers support an efficient way to distribute grouped data; this is called multicasting. In other cases, games may be limited to broadcast or point-to-point connections. Still, grouping can provide some efficiencies. Grouping can help a networked game scale up to include large numbers of players. This article covers logical ways to group data and the schemes used to implement grouped communications.

What Is a Grouping?

The first thing to consider is why you'd want to group data at all. If you're developing a one-on-one boxing game played on two computers, then chances are both players will need to know everything about one another; every move and every change will need to be exchanged. It's a very different situation, however, if you're building a large-scale space war scenario with lots of players. You may have a very high-performance spaceship and may be doing rolls in seven dimensions, but if I'm far away from the maneuvers, you just look like a dot to me. You're too far away for me to see your detailed maneuvers, so I don't need to know the

Using Groupings for Networked Gaming

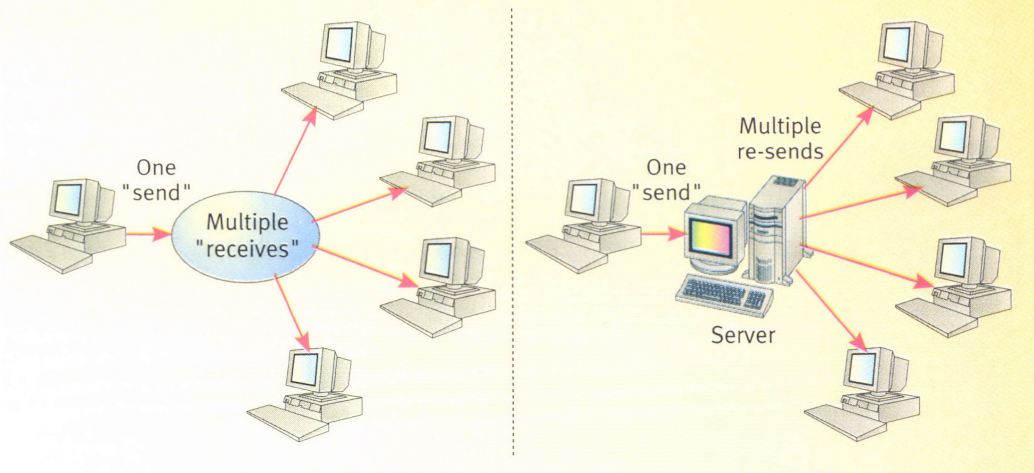


details of your motion. Furthermore, if neither one of us can see the "cloaked" alien spaceship hovering nearby, then there's really no need for him to send us his position at all until he becomes visible. The bottom line is that if you think about various players' data requirements, you can save a lot of bandwidth, network interrupt overhead, and CPU processing by only sending information to the players who need it. The trick is to make routing decisions without spending more cycles than you stand to save.

How Is Grouped Data Transmitted?

Before we get into the mechanics of creating data groups, let's look at how grouped data can be distributed among computers on a network. Ordinarily, each sender can communicate with other players on the network using either broadcast, one-to-everyone communications such as UDP/IP or via one-to-one links such as TCP/IP. For computer games, what you really want is one-to-many or many-to-one communications. Multicast-capable networks

FIGURE 1. Multicast transmission vs. server message exploding.



group in DirectPlay, any player within a session calls `IDirectPlay3::CreateGroup`. After a group is created, any of the other players in the session can join the group via the `IDirectPlay3::AddPlayerToGroup` method or leave via the `IDirectPlay3::DeletePlayerFromGroup` method. Any player can then send a message to all the other players in the group via a single call to `IDirectPlay3::Send`. Note that these interfaces don't change with the underlying communication mechanism of the service provider; rather, DirectPlay abstracts away whether a group is implemented as a physical multicast group, an exploder group, a broadcast that's filtered on the receiver end, or some other mechanism.

GROUPING ALLOWS NETWORKED GAMES TO ROUTE ESSENTIAL DATA AMONG THE PLAYERS WHILE MAKING EFFICIENT USE OF BOTH NETWORK AND CPU RESOURCES. *by JEFFREY ARONSON*

implement this capability at the physical level, using addressing schemes that allow a single transmitted message to be routed to many receivers while being ignored by others, even others on the same LAN. Even in environments where multicasting is not available, or in server-based game environments, the server can still effectively implement multicasting. A single transmission is reflected (or "exploded") back to multiple receivers, albeit with added latency. Figure 1 shows multicast transmission and its server/exploder equivalent.

The DirectPlay component of DirectX that provides an abstraction of one-to-many and many-to-one messaging is called, not surprisingly, group management. DirectPlay's group management methods provide the ability to define, join, leave, and send data to groups of players. To create a

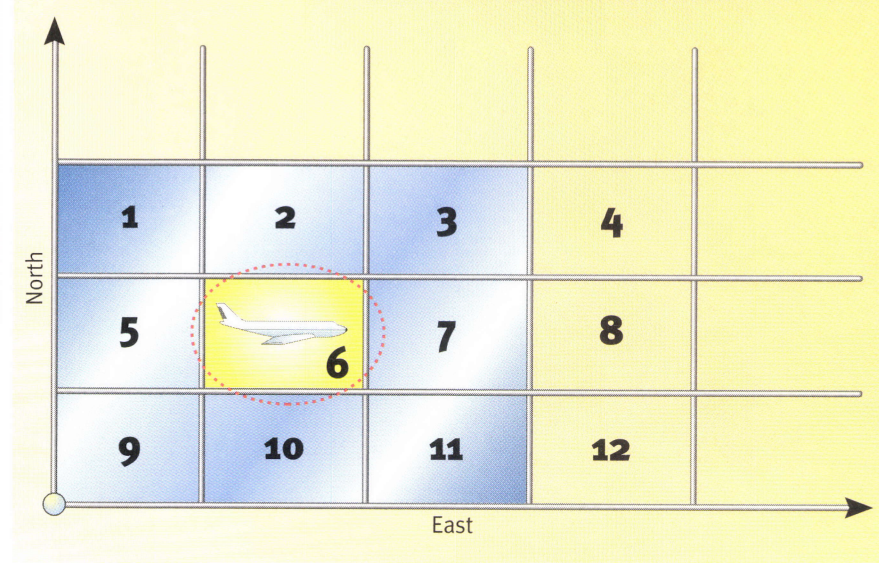
Assigning Data to Groups

Basic multicast technology doesn't say anything about what types of groupings to use, and indeed, in DirectPlay any player can join any group. So how do you make best use of groups? The approach to take in grouping data is to come up with some easily calculable measure for each message and to route the message to groupings based on that measure.

When looking at any piece of data, the game has basically three pieces of information that it can use to decide where to send it.

1. The data contents itself (such as the geographic location of an avatar, what team the producing entity belongs to, what

FIGURE 2. *Geographic grouping.*



markings it has, what entity type it is, what acceleration it has, and so on). This is called "data-based grouping."

2. The source of the data (that is, the place the data is coming from — either computer, site, player, or object). This is called "source-based grouping."
3. The destination of the data. In "recipient-based grouping," data is sent to a group comprised solely of those players who are to receive that information.

DATA-BASED GROUPING. Data-based grouping is the most intuitive form of creating groupings, and for many games, geographic sectorization is the most common form of data-based grouping. Under geographic sectorization, the virtual world is broken up into regions and data is grouped according to region. For example, in the scheme shown in Figure 2, 12 groups are used to segment the play area based on a rectangular North/East grid. Each entity knows what region it's in and sends its data to the corresponding group. Thus, the airplane shown in the figure sends data to Group 6. It's important to note that entities don't have to recheck their region location with every internal update (that is, every time they update their position). If you know that a single region is 100 miles or 20 light-years across, and you know how fast your vehicle can go, then you know the soonest you could possibly enter another region, and thus can postpone checking your region until

that time. The result is that geographic sectorization yields an inexpensive (because it's infrequently calculated) way of creating groups of data.

A player's computer would control the flow of data to it by joining only those groups it cares about. For example, as shown in Figure 2, an entity in Region 6 might care only about entities in its immediate area, in which case it would only need to listen to Group 6. If the entity has a larger area of interest, it might subscribe to adjacent groups as well, in this case adding all the groups shown in blue. Of course, there's a tradeoff in using simple groupings; data isn't perfectly filtered. Let's say the entity in Region 6 is interested in all other entities within a certain radius of its location, shown by the dashed circle around the entity. In this case, the entity still has to join groups covering the entire blue region and will get some extraneous information. In general, this is acceptable — the extra data just has to be filtered out upon receipt.

Geographic sectorization is just one example of data-based grouping. In practice, any of an object's attributes can be used to assign that object to a grouping. For example, my space war game might simply sort data into two groups: data for "cloaked" spaceships and data for "uncloaked" ships. Players who don't have the ability to see cloaked ships listen only to the uncloaked group and therefore don't waste time and bandwidth processing

packets for ships they can't see.

Grouping can also be based on data type. For example, in a war game, you might have separate groups for land vs. air vs. sea vehicles. Chances are that submarines don't care about individual infantrymen on the battlefield and can avoid having to ever see infantry data by using groupings.

SOURCE-BASED GROUPING. In source-based grouping, every source of data (where a source can be a LAN, a player, or even a particular entity) is assigned its own group, and every player then subscribes only to those sources that it cares about. Under DirectPlay, only one player per group would send to a group, though many players can issue `IDirectPlay3::AddPlayerToGroup` commands to listen to the group. The problem then becomes one of telling the destinations what information is being produced by each source and determining which groups to join?

In some cases, this problem is solved simply by well-known sources of particular data. For example, if there's one computer that provides weather and terrain information for the entire session, then all other players automatically know to listen to that player's group if they want to get weather updates. A more general solution lies in creating a special group, which carries very low frequency information about all entities or players in the game, to which everyone subscribes. Since all players subscribe to this low-fidelity data channel, they know rough information about all the entities in the game. They can then get more detail about the entities they really care about by subscribing to the groups representing the sources that they want to hear. One good thing about this scheme is that the number of groups used scales linearly with the number of sources. In principle, the "source" can be anything, including a single entity or avatar. A side benefit of this approach is that the low-fidelity group automatically gives each player enough data to create a rough, "radar-screen" picture of the entire play space.

RECIPIENT-BASED GROUPING. Finally, we come to recipient-based grouping. This scheme is something of an inverse of source-based grouping in that in this scheme groups are created on the basis of recipients with common interests.

trueSpace3

a new
way
of
creating

Create, animate and render in 3D like never before.

Sculpt  organic models using **metaballs** and

an industry first, **plastiform**.  Breathe life into

your models with the most intuitive **forward dynamics**

and **inverse kinematics** features around. Paint directly on

your models in the same workspace as final rendering or VRML browsing.

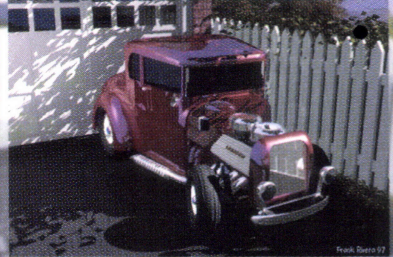
Don't let your creativity be limited by your budget, get affordable power
with trueSpace3! **30-DAY MONEY BACK GUARANTEE**

Try trueSpace3 risk-free for 30 days! Or download a FREE trial version of trueSpace3. Visit our website for details. You'll find additional product information, a trueSpace art gallery, tips, and contests! Experience affordable power today. Available for Windows 95 and NT 4.0

1.800.351.7620
www.caligari.com
email: sales@caligari.com

3D Tools for Creative Minds

Illustration & Animation



"I can stay in trueSpace3 from start to finish of a project. It is so intuitive that I have more time to create." -Frank Rivera, LOGICbit Online 3d Magazine

Game Design



"trueSpace is a terrific graphics package for breaking into the game business." -Tom Marlin, President of Marlin Studios, developers of "The Roswell Omen"

Web Design



"trueSpace's VRML support eliminated most of the hand coding required for the 14 worlds of Yahoo!3D." -Jef Hancock, Yahoo!3D Artist

For this scheme to work, each sender needs to know which players might be interested in each message it sends out. Once a list of interested parties is known, the data can be addressed to the groups that represent just those destinations.

As an example, consider a situation similar to that discussed under geographic grouping. Once again, each player is interested in other nearby entities. This time, though, each player must send a message to all the other computers in the game saying something along the lines of "I'm at location (x,y,z) and want to know about anyone within one light-year of me. By the way, I listen to multicast Group 10." This interest expression tells the other computers in the game what this player's data needs are and what communications channel to use. Then, each other computer, when it's about to send out information about its entities, must look at the interest expressions from all the other players on the network and transmit the data over the proper multicast groups.

This scheme carries with it some extra communications overhead in that the data requirements of every player must be transmitted to every other player during the exercise. Fortunately, these interest expressions change rather slowly, so they don't result in a large amount of information flowing between hosts, as is the case with dynamic game data. Recipient-based grouping can also use up the largest number of groups of all the possible schemes, if you're interested in creating groups for each possible combination of recipients. Recipient-based grouping can be expensive to implement in terms of CPU cycles on the sending side, but unlike either of the other schemes, it can offer perfect segmentation of data (you receive only what you need, with no extra information).

Dynamic Grouping

All of the grouping schemes described previously are based on static definitions of groups. For example, in the geographic sectorization scheme, the boundaries of Sector 1, and hence, the definition of Group 1, are defined before the game starts running. This is the easiest way to imple-

ment groups, since group definitions, and consequently, choices about where to send or listen, can be hard-coded into the game.

Sometimes, however, static group definitions don't work out. Suppose in my spaceship game a heated battle is taking place in Sector 6. The remainder of space may be relatively empty, but there's a lot of activity in Group 6. In this case, using groups doesn't buy you much in the way of performance, since most of the players wind up sending and receiving Group 6 data. The solution in this case would be to create dynamic group definitions. With dynamic groupings, each session would designate a player to serve as a group server for the session (under DirectPlay, this role could be filled by the session host). The group server monitors or infers traffic flow in the various groups and can dynamically redefine groups. In the spaceship game, for example, a group server might decide to break up Sector 6 into four smaller sectors. It might add new groups, sending out the new group definitions to each player. If the number of groups available to a session is limited, perhaps by the available number of physical multicast groups, then it might redefine existing groups (for example, by combining Sectors 4 and 8) to free up groups. Figure 3 shows what the geographically sectorized space of Figure 2 might look like after being dynamically adjusted. The bottom line is that with dynamic groups,

each player's software must listen to group definitions as broadcast from the group server and modify its group memberships accordingly.

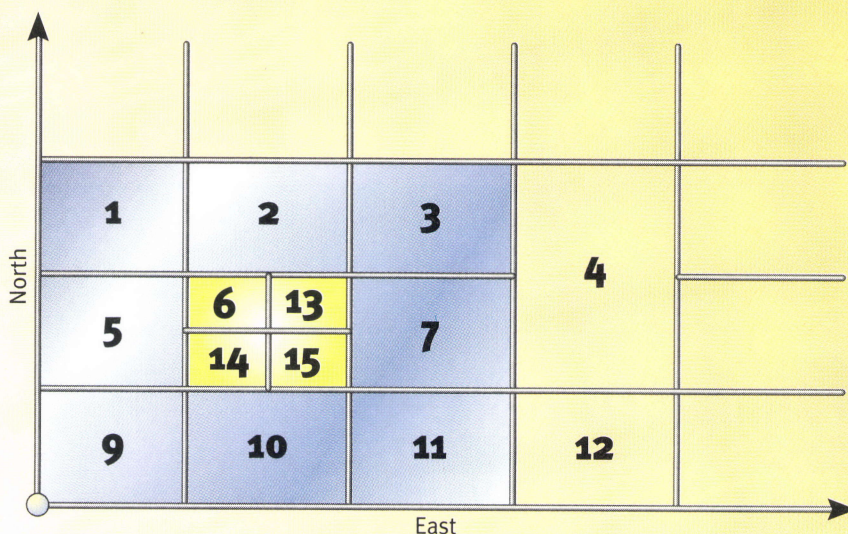
Two Grouping Gotchas

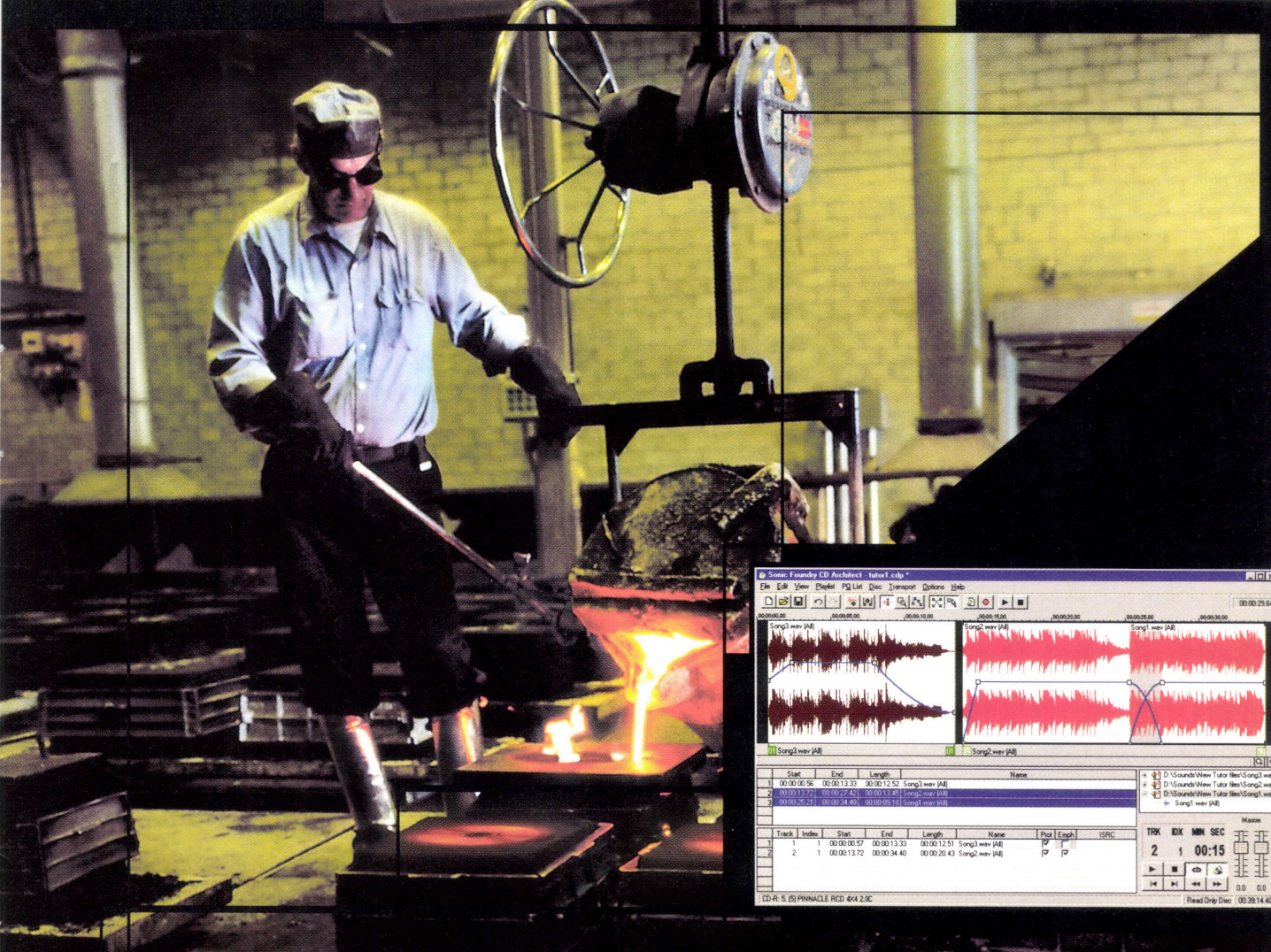
There are a couple of things to look out for when you use groupings to communicate among networked games. The first challenge is to understand the underlying transport mechanism used by the service provider to implement grouped communications. Certain implementations, such as server exploders, impose additional communications overhead on grouped communications. In other cases, such as poorly implemented multicast drivers, the use of large numbers of groups can actually cause a decrease in performance. Unfortunately, most game developers have no control over multicast driver performance, as network drivers are part of the operating system.

One important thing to look out for is the join/leave delay for groups. In general, group creation, joins, and deletes don't happen instantaneously (in fact, group join/leave times are unpredictable, but are generally measured in seconds); games must be prepared to forecast their data distribution needs sufficiently far in advance to allow time for joining and leaving groups.

The other major problem you'll face with grouping is how to bring

FIGURE 3. Dynamic adjustment of geographic grouping.





Build Audio CDs by the Book

Design professional audio CDs to Red Book spec with Sonic Foundry's new CD Architect. Tracks in the PQ List can be assembled with regions from single or multiple sound files. Full PQ code editing allows even the most sophisticated mixes and crossfades between tracks. Used as either a stand-alone editor, or as a Sound Forge 4.0 plug-in - CD Architect is ideal for building audio CDs on Windows 95 and Windows NT.

- Supports 99 tracks per disc (99 sub-indices per track)
- Allows single or multi-file playlisting
- Verifies PQ list for Red Book compatibility
- Fully supports PQ code editing including track and index positions and pause times
- Includes dozens of audio processes, tools and effects with multiple levels of undo/redo
- Provides independent-channel master volume faders and adjustable envelope controls

- Burns disk-at-once premasters suitable for creating glass masters
- Generates printable cue sheets
- Previews multiple tracks or ranges of audio before extraction
- Auto-detects many of the CD-ROM drives, CD-recorders, and auto-loaders it supports

CD Architect includes native versions for Windows 95, Windows NT-compatible (x86, Pentium, Alpha systems) on a single CD-ROM.

CD Architect

design and build audio CDs with speed and precision
1 800 57 SONIC or www.sonicfoundry.com



100 South Baldwin, Suite 204, Madison, WI 53703 Tel: (608) 256 3133, Fax: (608) 256 7300, CompuServe: 74774,1340 or GO SONIC, Internet: sales@sonicfoundry.com. Sonic Foundry and Sound Forge are registered trademarks of Sonic Foundry, Inc. Other products mentioned are trademarks or registered trademarks of their respective manufacturers.

players up to speed when they join new groups. In a grouped, multicasted world, each player knows only a subset of the globally available data. When a player joins a new group (such as, when a spaceship flies into a new sector), some anomalous things can happen. If the player doesn't immediately get all the information about all the data in that group (such as, the identities and locations of all the other ships in that sector), then he's essentially flying blind for a period; our ship can be blasted by another ship that we've never even seen! This is another case where it's important to join groups well in advance of when you'll actually need their data. If, on the other hand, the game implements a mechanism to bring a new group member up to speed quickly, then a player who joins a busy group may immediately be flooded with information, resulting in packet loss or poor application performance. Any protocol to bring group-joiners up to speed must care-

fully avoid this problem. This is another situation where a low-fidelity data group (as described under geographic sectorization) can be useful; if all players listen to the low-frequency broadcasted updates on the low-fidelity group, then they'll have at least some idea of what to expect when entering a new sector.

Putting It All Together

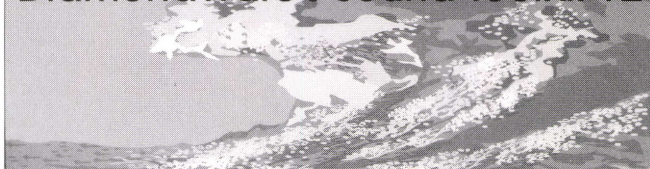
Managing data distribution across networked games is not an easy problem. Simple data distribution schemes don't scale well, limiting the potential size of games. More complex schemes, such as the various types of data grouping, require more thought on the part of programmer, but have been shown to greatly increase the potential scale of games. In the military simulation world, the increased network and processor efficiencies resulting from multicasting/grouping have yielded an order of magnitude increase in the number of entities sup-

ported when compared to broadcast-based Distributed Interactive Simulation protocols. Current game SDKs, such as Microsoft DirectPlay, offer group facilities but don't provide any support for group definition or use. Any game developer looking to create scalable networked games needs to consider efficient data distribution. The good news is that combining SDK grouping functions with group definition concepts such as those described in this article can yield games that are suitable for large-scale play by large numbers of distributed gamers. ■

The author would like to thank Dr. Edward Tiberius Powell and Larry Mellon, whose pioneering designs for the JPSP and STOW projects form the basis for many of the ideas presented in this article.

Jesse Aronson is a software architect at Science Applications International Corp., where he designs war gaming simulations covering thousands of entities running on hundreds of computers across dozens of sites. He can be reached at jesse.s.aronson@cpm.saic.com.

DiamondWare's Sound ToolKit v2



The Next Wave in Interactive Audio

So you're making an internet game, and putting it on a CD. You're supporting chat, which means half- or full-duplex recording... and voice compression... and voice-disguising DSP. The graphics look 3D, so the audio has to sound 3D. You have room on the CD, so you might as well use it for a studio-quality music track.

Or maybe you're building an interactive story. There's so much audio content, that you need 4:1 compression. You want different compressors for speech and SFX. It's essential that the music change with the context of the story, no dumb loops. This rules out conventional digital audio. But, young JS Bach said he'll only author for MIDI if he can record the instrument samples.

You could spend lots of time and money developing all this technology yourself. Doesn't it make sense to buy into a ready-made solution instead?



www.dw.com

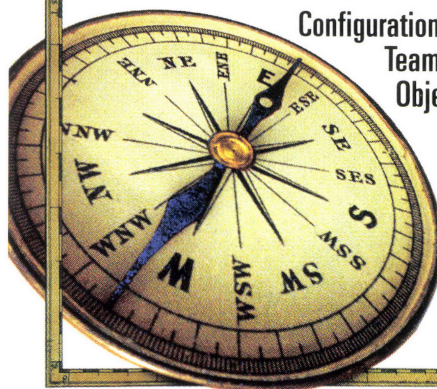
Developed by DiamondWare
Distributed by The Coriolis Group

(800) 410-0192 (orders)
(602) 917-3474 (tech)
(602) 917-5973 (fax)

Direction for Corporate Developers

Find your way to increased productivity with **Software Development!**

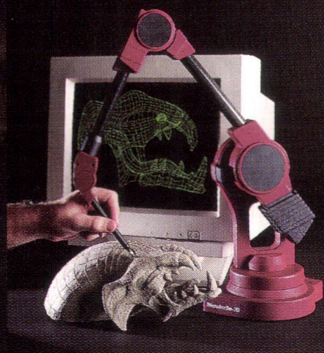
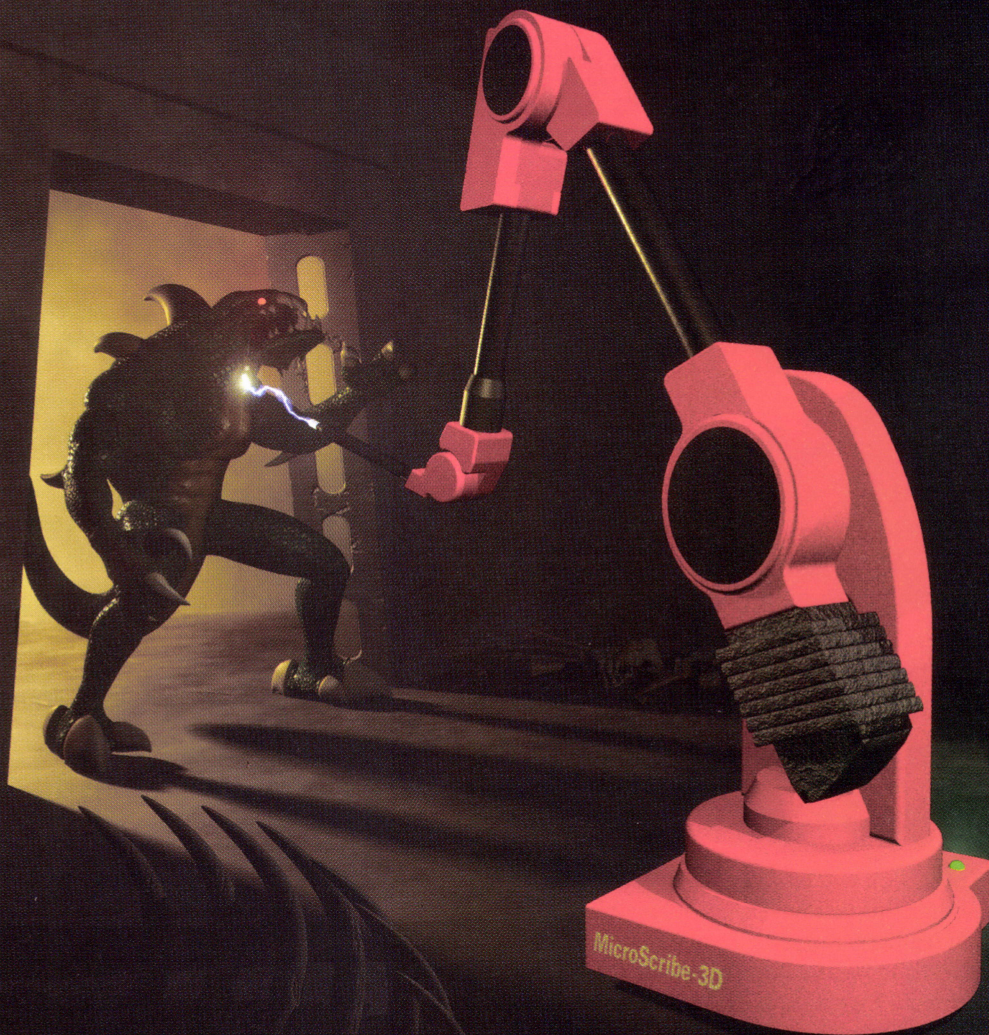
www.sdmagazine.com



Configuration management
Team development
Object technology
Intranets
Testing
Java
...and more

SOFTWARE Development

Prepare to meet your maker!



MicroScribe is an advanced 3D digitizing tool that empowers you with the ability to quickly and accurately capture complex 3D models from physical objects by tracing over their surfaces.

The MicroScribe-3D is a desktop system that easily connects to your serial port and runs on PC, Mac, and SGI platforms. MicroScribe is compatible with all your favorite 3D graphics, animation, and CAD software. Models created by MicroScribe can be exported in IGES, DXF, and other standard file formats. For details, call Immersion Corporation at 408-467-1900 or 800-893-1160 or visit our Web site at www.immerse.com.

MicroScribe-3D™
... a new dimension in desktop digitizing



**Immersion
Corporation**

"The Guardian"™ & © Hanna-Barbera

Using a Base-Root System For Animated Characters

50



So, you're programming a 3D game? Then you've left the world of sprites forever. You'll have more freedom, better animation, and fewer headaches than you had in the sprite world. Your rent will go down and chocolate will taste better. Random people will stop you in the streets just to say, "Have a nice day." Your laundry will do itself. Cops will no

longer pull you over. You will be given honorary degrees in fields you never heard of, and will regularly receive invitations to the White House. But first, you must write an easy-to-use, flexible animation system.

For your first shot at 3D animation, you choose to animate a guy walking. So you ask your favorite 3D animator to give you a walk cycle — something simple like a guy walking five feet or so. And, animator buddy, please make the last frame lead into the first frame so the whole thing can be looped.

This first shot at a walk cycle works pretty well. You rip the animation with your tools, put it up in your 3D animation engine, and the guy walks five feet. But, at the end of the five feet, the animation suddenly pops back to the beginning. Walk five feet, pop back, over and over. Obviously, this is the animator's fault, right? The character should have been created to walk in place.

You complain to the animator, explaining the mistake. The animator just shrugs and removes all of the forward translation from the animation. This is fine by you, because you want control over your character's velocity anyway. You put the animation up in your engine, and the guy walks in place. Adding a simple forward velocity gets him moving, and with a little bit of adjustment, the feet don't slide very much. But something is still not quite right. Your animated character moves exactly like those boxy microwave oven, color TV, and refrigerator movers in Dire Straits' old "Money for Nothing" video. The walk is completely unnatural and robotic. It looks, well, computer-generated.

So, maybe the animator was correct to give the character forward movement. Animators should be able to make a walk look the way it is supposed to look, and programmers

should be able to handle it. From the programmer's point of view, it would be nice to simplify things, including complex, motion-captured animations. A nice way to do this is by introducing a "base coordinate system." In addition, "linear velocity extraction" can solve problem such as our walk animation popping back five feet.

First, let's define some standards. When animators create walking creatures (hand-animated or motion-captured), they'll likely work with a floor as their reference — I'll assume this floor is just a display of the plane $z=0$ (Figure 1). The world is a right-handed coordinate system where the positive z axis is "up." From the animator's point of view, the creature's root node is at the top of its skeleton hierarchy; move the root and the whole creature moves in the world. This root is usually positioned at the creature's center of gravity (for a human, the

root node would probably be the joint between the torso and the hips). For a nonexploding, nonstretching, skeleton-based creature, this root node is the only part of the animation that will have changing translations in x, y, and z, in addition to rotations. Other segments will have translations from their parent, but the translations will stay constant for all animation frames. Note that this means that in your animation data, you only need to store one translation keyframe for all segments below the root.

have a point defined for each frame that anchors the frames. Velocities for the character can be applied to one point, and as long as each frame's hot spot is aligned with this point, the animations will appear correct. Our base coordinate system "hot spot" can be thought of as an approximate projection of the root onto the 2D plane of movement. It will always be approximate, as we want to be able to move the base in simplistic ways while allowing the root to deal with the natural movement of the center of gravity.

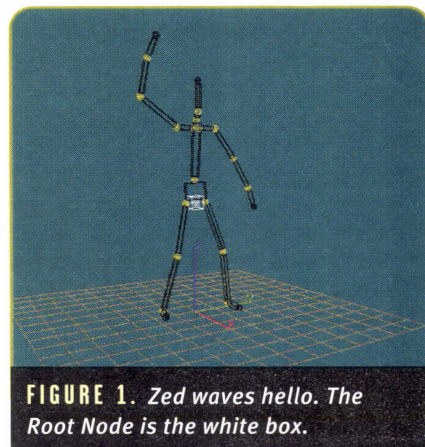


FIGURE 1. Zed waves hello. The Root Node is the white box.

ANIMATORS SHOULD BE ABLE TO MAKE A WALK

LOOK THE WAY IT IS SUPPOSED TO LOOK,

AND PROGRAMMERS SHOULD

BE ABLE TO HANDLE IT *by Scott Corley*

Consider the motion of the root for a character doing something as simple as walking. There will be a forward translation as the character moves forward, but the root doesn't move forward at a constant velocity in a natural-looking walk. There will also be translations to the left and right, as well as up and down. If the animation was motion captured, the root will move in all sorts of unpredictable (but natural-looking) ways. After spending so much time and money getting the animation just right, the last thing we want to do is strip out this natural movement.

I'd like to introduce the character that will be used in all examples. Call him Zed. He's a human-like figure with an 18-segment skeleton, pictured in Figure 1. Zed's root is attached to his hips. "Zed's World" is the world coordinate system in a fictional game. And Zed's capabilities (or lack thereof) will determine whether Zed gets published or canceled.

Introducing the Base Coordinate System

The base coordinate system we would like to use serves the same function as a "hot spot" on a 2D sprite. For sprites, it's convenient to

Our base coordinate system will be the parent of Zed's entire hierarchy, so it is the parent of Zed's root. The base will be used to translate Zed or to change his facing (rotation about his z axis). The root will be controlled by the animation data at all times, so we don't need to worry about it. From a programming standpoint, we can simply view Zed's general position and facing as the position and facing of the base.

When preparing Zed for the base coordinate system, the animator doesn't necessarily need to add another level to his skeleton hierarchy. However, there are a few standards that should be followed. A standard walk animation should start with Zed's root squarely over the origin (as in, x and y are zero) and the z position of the root set so that Zed's feet line up correctly with the floor (the plane $Z=0$). Zed should be facing down the positive x axis. Zed's left side will then point down the positive y axis. Whenever possible, the animation should end with Zed still facing down the positive x axis.

When these animation standards are followed, we can add our base-coordinate system in Zed's world. The base-coordinate system will always have a z translation of 0, and the x and y translation will move the character around the floor. The z axis of the base coordi-

nate system will always be aligned with the z axis of the world, and the orientation of the x and y axes (the rotation about z) will make Zed turn to face his foes. Zed's root z position is controlled by the animation data, so if the feet look correct on the artist's workstation, the root translation will ensure that the feet are planted firmly in Zed's world. In Figure 2, the base is on the floor (the red and green axes), and the z translation of the root (the white box) maintains the foot's relationship to the floor.

Now Zed can be controlled in a very sane manner with the base coordinate system. Setting the x and y translations of the base to zero will put Zed square in the center of his world. Zed's feet will be on the floor just as the animator intended. In addition, setting Zed's facing (rotation about the z axis) to zero will have him facing straight down the positive x axis. You can, at any time, examine the base translation and facing to determine where Zed is and which way he is facing.

See Zed Sit, See Zed Jog

Let's kick back with a few examples before delving any further. Zed now has a base coordinate system that is always in the same plane as the

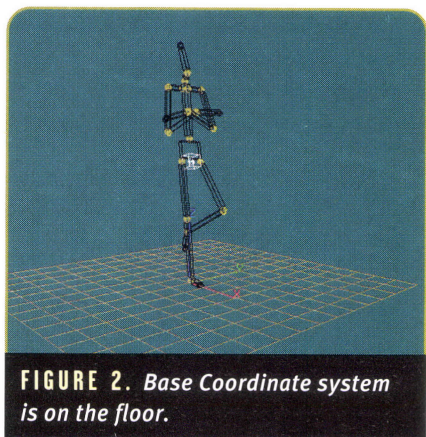


FIGURE 2. Base Coordinate system is on the floor.

floor, even if he's jumping or falling down. We can move Zed around the floor by moving his base, allowing us to apply a simple linear velocity at the programming level. The root, which we now know is somewhere just above the base, gets its translation and rotation from the animation keyframes so that it can swing and sway naturally, just as the animator intended. Note that if the programmer were controlling the root directly, Zed would have no swing to his step, and his forward motion would be completely linear (money for nothing...).

The simplest example is of Zed standing still after running away from some bad guys. The animator wants Zed to bend his knees and heave as he breathes in fresh atmosphere. This is fine, because the base coordinate system doesn't move at all. As far as the base is concerned, Zed is standing still. The root moves according to the animation data, but it is always generally over the base.

What if Zed is so tired that that he has to sit down for a minute? That's fine. Our base doesn't move. The animation keyframes take care of lowering Zed's root so he can take a little rest.

Later, Zed wants to do a little jogging. This is simple, too. We just play Zed's jog animation, and we move his base forward at the appropriate velocity (stored with the animation). Zed's jog is sure to be a bit jerky, but as far as our base is concerned, it's simple linear motion.

Zed's world would not be complete without some jagged spikes to leap over. In the case of a canned jump, the animator can supply us with all of the z translation we need. In the case of motion capture, the z translation will

be perfect, so we don't want to lose that. The base coordinate still stays planted in the x-y plane. The root's animation keyframes take care of animating Zed's z translation.

Linear Velocity Extraction

Walking and running are two activities that Zed will do often. The natural way to animate these actions is to include the forward translation in the animation. For motion-captured actions, this translation is already there, and it's exact. By now you are probably wondering what happens between animations. If Zed's run cycle makes him travel ten feet, won't his base be left behind? The answer is yes, but this is where linear velocity extraction (LVE) comes in.

LVE is a processing step that needs to happen before Zed is introduced to his world. A full explanation of LVE is the topic of another article, but the concept is simple. If our animation-processing tool is presented with an animation that travels a significant distance, a velocity needs to be extracted from it — later we can add this velocity back into Zed's world. If an animation travels nine feet along the positive x axis, and the animation is one second long, we need to extract a linear velocity of nine feet per second and store this velocity with the animation. Zed's animation is created at 30 frames per second, so for this example animation he's travelling at nine feet per 30 frames, or roughly 0.3 feet per frame. This distance must be subtracted from each frame's translation when it's processed by the programmer's conversion tool. The animation will look as if it's running in place until we add in the forward velocity.

After LVE has been applied to an animation, the rules for the base coordinate system will all apply. The root will always be approximately above the base. In addition, we haven't lost any of the subtleties of Zed's motion. His center of gravity will still have a quirky, natural-looking motion to it, but at the programming level, all we've done is to move Zed's base along at 0.3 feet per frame. The result will look exactly as it did back at the artist's station, and we programmers are happy

because from our point of view, we're just moving a point around the floor.

Horizontal motion (along Zed's y axis) is handled the same way. The velocities stored with each animation contain an x and a y component, so a sideways strafe is just as easy as forward motion. In fact, motion in any direction on our floor is handled easily by these velocities.

The velocity extraction is only applied to the x and y axes, however. Simple z movements, like the canned jump described earlier, are kept in the animation and need no attention from us. We don't need to worry about gravity or the upward force of the jump. In Figure 3, we just let the jump animation play out, and our base stays at z=0 while the root flies through the air. Any x and y motion for the jump is taken care of by our extracted velocities.

Controlling the Animation

Anybody with any sense of game play knows that you can't use canned moves in all cases. What if you want to give the player some control over his jump? What if Zed can transform himself into a giant superball and bounce around?

The simplest way to add more control to a canned animation, such as a jump (Figure 3), is to stick with the system described thus far. The distance of the jump can be varied by altering the velocity applied to the base. If the jump needs to go 10% farther than normal, then increase the applied velocity by 10%. If the desired landing

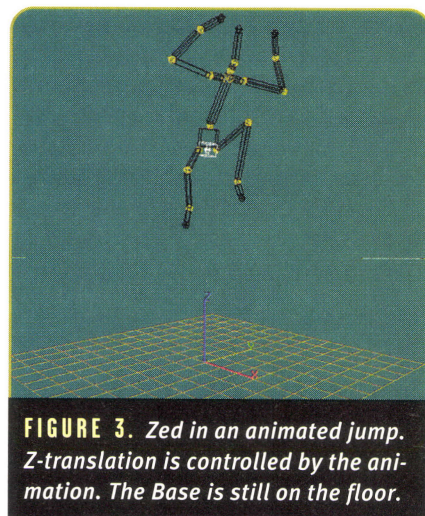


FIGURE 3. Zed in an animated jump. Z-translation is controlled by the animation. The Base is still on the floor.

IF **WEB** PROFESSIONALS HAD **REGULAR** JOBS, MAYBE



What they said about
Web '97 in San Francisco:

"Nowhere else can I find
such a concentration
of practical, how-to
information on web design
and development."

Wallace Kowrach
Manager
Internet Communications AAA

"Defined my role as a
web designer."

Lorelei Linder
Web Designer
Dell Computer Corporation

SPONSORED BY:

WEBTechniques

Web Review

CO-SPONSORED BY:

Microsoft

macromedia

WE WOULD HAVE **CREATED** A **REGULAR** **EVENT.**

WEB DESIGN & DEVELOPMENT '97

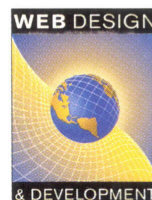
CONFERENCE: SEPTEMBER 29–OCTOBER 3, 1997

EXHIBITION: OCTOBER 1–3, 1997

WASHINGTON CONVENTION CENTER, WASHINGTON, D.C.

Make no mistake about it - this Web stuff is serious business. To stay on top, you need solid training from the best minds in the business. From the fundamentals of site usability to cutting-edge programming techniques, Web Design & Development has set the standard for giving Web professionals what they need to succeed. Practical training, from leading experts in intranet strategies; Java vs. ActiveX; useful interface design; dynamic animation; secure transactions; marketing techniques and more. Visionary keynotes. Peer networking and brainstorming. And an exhibition of powerful tools from the hottest companies around. Join us for the most intense gathering of Web brilliance the East Coast has ever seen. Because when you're serious about your job, there's only one place to go.

BIG. VERY BIG.



un Miller Freeman
A Times News & Media publication

w w w . w e b 9 7 . c o m

Feeling nostalgic? Contact us by phone at 415.905.2702 or by e-mail at web97@mfi.com

LISTING 1. When Zed reaches the end of the initial leap-tuck animation, the base is hoisted up by setting it to have the same x, y, and z position as the root.

```
void ZedGuy::HoistBase(void)
{
    CurrentRootPosition=WorldRootPosition(); // Get XYZ translation of Root
    SetBasePosition(CurrentRootPosition); // Make Base position match Root position
}
```

position is known, the velocity required to get Zed to that point can be calculated and applied to the animation. A variance of more than 50% in

the base is hoisted up by setting it to have the same x, y, and z position as the root (Listing 1).

Zed is still controlled by translating

y axes (rotation about z) controls where Zed is facing. It also controls which direction he walks or runs, so the velocity that we add back to the base translation must be rotated to match the base facing.

The base coordinate system is always approximately under Zed's root, and the base is always facing approximately "forward," as far as Zed is concerned. What happens when Zed's animation causes him to face a different direction? The answer is that at the end of the animation, the direction that the base faces no longer represents "forward" in terms of Zed. We have to synchronize the base with Zed's root at the end of the animation.

Synchronizing the base facing with the root facing only needs to be done in animations where Zed intentionally faces somewhere new in the world at the end of the animation. A good example is a jump with a half twist, where Zed jumps into the air facing forward and lands facing backward. If we start this animation with Zed facing down the positive x axis, Zed's base will still face down the positive x axis at the end of the animation, but his root (controlled by the animation) will point down the negative x axis. To put it another way, Zed's nose normally points in the same general direction as the base; at the end of this animation, the nose and the base point in opposite directions. The half twist was intentional in this animation, and we want to make it "stick," so we have to force the base to face the same direction as the root.

How you implement base-root synchronization depends on your animation system. If no interpolation is being used, the synchronization is straightforward. If you are interpolating between animations, however, a new animation frame must be generated at the root for proper interpolation. Listing 2 shows the *JumpWithTwist* maneuver in detail.

What if you want to give the player some control over his jump? What if Zed can transform himself into a giant superball and bounce around?

distance may start to look strange, but with two or three different jump animations, a wide range of jumping distances can be covered.

Controlling the height of a jump in this way is not as easy. The z position of the animation is controlled entirely by the animation frames, and Zed won't follow a nice parabolic arc until his feet have left the floor. If the "flying" portion of the animation is separated from the initial jump, however, the jump height can be controlled by scaling the z translation values of the root animation. This must be done carefully, as changing the scaling too quickly during an animation will make the jump look unnatural. The final scale of the z translation should make its way back toward 1.0 (the original scale) so that the root will line up correctly with the landing animation. If the piece of floor being landed on is the same one that our base currently occupies, the landing is simple because the animation takes care of it. The feet will hit the floor solidly without any intervention on the programmer's part.

When full control is needed over Zed's root, it's time to hoist up the base. Let's look at Zed's superball move. When Zed transforms into a giant superball, he does so by leaping into a tuck position. From the tuck, Zed holds his knees, spins really fast about his y axis, and bounces when he hits the floor. When Zed reaches the end of the initial leap-tuck animation,

his base, but the root translations for all superball animations will be zero (that is, Zed is animated spinning around the origin). Zed's x, y, and z position can now all be controlled completely algorithmically, but we have to worry about part of Zed possibly sticking through the floor. Algorithmically controlled animations are a special case, and collision detection must be performed to make certain Zed bounces when he should. When it's time to transition back into a landing animation, we should wait until Zed's hoisted base position aligns with the starting root position of the landing animation. When the landing animation starts, we unhoist the root by setting Zed's base z translation back to zero and the animation's root z position takes care of the actual landing.

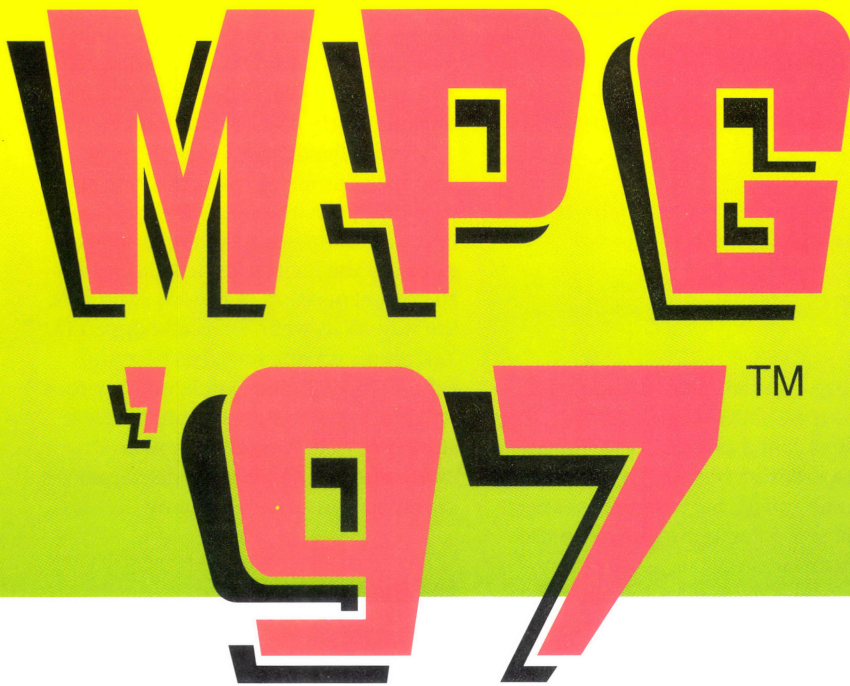
Zed's Facing

Now that the base system is up and running, Zed can be made to run, jump, or do anything else in any direction. The orientation of the base x and

LISTING 2. The *JumpWithTwist* maneuver.

```
// Zed's Base Facing starts out at zero
Zed->PlayAnimation(JumpWithTwist);
// Zed's Base Facing is still zero after JumpWithTwist, but the Root Facing is "backwards".
Zed->SetNextAnimation(StandReady);
// StandReady will set the Root back to "forwards," causing Zed to flip 180 degrees
```

Learn How to Profit from the Booming Internet-Based Gaming Market



The Multi-Player Online Gaming Conference of the Year!

September 15-17, 1997 • Pan Pacific Hotel • San Francisco

Meet the Industry Leaders of Online Multi-Player Gaming and Learn How to:

- ▶ Develop sources of revenue for gaming online (Who's making money and how)
- ▶ Take advantage of the growth rates and direction of online gaming
- ▶ Create a multi-player gaming community to increase awareness and profits
- ▶ Leverage innovative methods of marketing multi-player gaming
- ▶ Create strategic partnerships and alliances by networking with industry leaders
- ▶ Make it easier for your customers to get online and experience your games

HEY!! Wanna be the biggest and baddest player in the hyper-competitive gaming market? Whether you are a lean and mean outfit that is super creative or a scary gaming behemoth ready to put the hurt on the little guy **DON'T MISS OUT! THIS IS THE GAMING NETWORKING EVENT OF THE YEAR.**

PLUS, Attend the 2 "Info-Packed" Post Conference Workshops on September 17th

Workshop A: *Business Strategy Section: Develop a Strategic Revenue Plan for Your Online Games*

Workshop B: *Technical Section: Improving Problems of Latency For Your Games Online*

Presented by:



Sponsored By:



Publishers of:



Organizers of:



HOLY COW!

Check Out the Gaming Experts Who Will be Speaking at this Event:

John Taylor, President,
KESMAI CORPORATION

Paul Matteucci, President,
MPATH INTERACTIVE

Jeff Leibowitz, President,
ENGAGE GAMES ONLINE

Mike Moniz, President,
VR-1

Mike Wilson, CEO,
ION STORM

Jon Grande, Executive Producer,
INTERNET GAMING ZONE, MICROSOFT

Chris Lombardi, Editorial Director,
TOTAL ENTERTAINMENT NETWORK

Chad Little, President & CEO,
SANDBOX ENTERTAINMENT CORPORATION

Neil Harris, Executive Vice President, **SIMUTRONICS**



For More Information, Call Us Toll Free at 1-800-882-8684, Fax 201-256-0205

Send E-Mail to: info@iqpc.com

Visit our homepage at: <http://www.iqpc.com>

LISTING 3. The animation setup with the EnableBaseSyncWithRoot call.

```
// Zed's Base Facing starts out at zero.
// Notify the animation system that the next animation doesn't end with Zed Facing forward
Zed->EnableBaseSyncWithRoot();
Zed->PlayAnimation(JumpWithTwist);
// After JumpWithTwist ends, the animation system will synchronize the Base Facing to match
// the Root Facing. StandReady will then face down the negative X axis as expected
Zed->SetNextAnimation(StandReady);
// Zed is now standing facing down the negative X axis.
```

The jump-with-a-half-twist animation sequence should illustrate the problem we need to solve. The answer is to notify the animation system that `JumpWithTwist` requires a base synchronization with the root. After `JumpWithTwist` has finished, the animation system must determine the x-y facing of the root and set the base to match its direction. If the next frame

Listing 4 shows some pseudo code for a routine called `BaseSyncWithRoot`. This routine is called as soon as we detect that `JumpWithTwist` has ended (we know it should be called, because `EnableBaseSyncWithRoot` sets a flag telling us to do the synchronization).

One of the basic rules for our animations was that each should start and end with our character facing down the

of the more complex situations that you may need to deal with, but here's an example to get you thinking about another fairly common situation.

The `BaseSyncWithRoot` example only solves our problem when the base and root don't face the same direction at the end of an animation. What happens if not only the facing doesn't match, but the actual position of the base and root are way off as well? This can happen if Zed pivots on one foot. His base and root facing will not match, and the root will end a few feet away from where it started as well. A more complex `BaseSyncWithRoot` can handle discrepancies in translation as well as facing.

And More

There are many situations that you'll have to deal with in your 3D animation system that won't be completely obvious when you start writing code. Many of the bits and pieces of a full system have been brushed over here — the important thing to keep in mind is a design that is flexible enough to handle new problems. Starting with a good design can mean the difference between a publisher telling you that Zed will get some good marketing before Christmas, and a publisher telling you, "Zed's dead, baby. Zed's dead."

The author would like to thank Harold Merrill for working up the fine illustrations of Zed that accompany this article.

Scott Corley is vice president of software development at High Voltage Software Inc. He's getting married this August to a wonderful girl who actually thinks game programming is cool. He can be reached at scottcy@ripco.com.

56

The base-root system will make many animation problems easier to solve — you'll be able to say "We can do that" far more often than you say "No way."

displayed is the first frame of the `StandReady` animation, the `StandReady` will face in the direction that we expect. However, if we need to interpolate between `JumpWithTwist` and `StandReady`, we need one more step. Listing 3 shows how the animation setup looks with our new `EnableBaseSyncWithRoot` call.

If we need to interpolate between `JumpWithTwist` and `StandReady`, the last frame of `JumpWithTwist` must be modified to offset the change we make to the base facing. Imagine what would happen if we didn't account for this change. `JumpWithTwist` would end, we would set the base to face down the negative x axis, and then we would interpolate from the last frame of `JumpWithTwist`. The problem is that the orientation of the root in the last frame of `JumpWithTwist` doesn't match the orientation of the base, so we have another 180 degree flip problem. The solution is to copy the last frame of `JumpWithTwist` (for the root only) and set the x-y root to face the same direction as the new base facing. Now the last frame of `JumpWithTwist` can be played with the new, properly adjusted base facing, and Zed's orientation will be correct.

positive x axis. The `BaseSyncWithRoot` adjustment takes care of situations when an animation starts out according to our rules, but must end with the character facing a different direction.

Are All the Problems Solved?

The system described in this article can handle animations with or without velocities, and will even handles a case in which we don't follow our own rules. Unfortunately, it would take another article to describe some

LISTING 4. Using BaseSyncWithRoot to synchronize the base and the root.

```
void ZedGuy::BaseSyncWithRoot(void)
{
    NewBaseFacing = WorldRootXYFacing(); // determine the orientation of the Root
                                         // in terms of the world
    SetBaseFacing(NewBaseFacing); // Set Base Facing to this new Facing
    NewInterpolationFrame=CurrentFrame; // Get a copy of the current animation frame
    NewInterpolationFrame->SetXYFacing(NewBaseFacing);
    // If we now interpolate from NewInterpolationFrame, the character's Root Facing
    // will match in the transition from the old animation to the new animation.
}
```

ECTS 97

Europe's interactive entertainment expo
7-9 September 1997, Olympia, London



Speed past the queues

ECTS 97 will be bigger than ever, with two halls of exhibits showcasing Europe's key interactive entertainment companies. As the interactive world's European focus, ECTS 97 gives you an easy high-speed route to all the newest products and technologies.

This year there are more reasons than ever to visit ECTS 97

- Over 150 of the world's best known companies; Nintendo, Sony, Sega, Eidos, Electronic Arts, Microsoft, Intel, Virgin Interactive Entertainment, MicroProse plus many more
- Hundreds of innovative new product & technology launches
- Extended opening hours, giving you more time to make valuable contacts and sign up new deals
- Increased CTW Premier Club facilities, the affordable business base for your visit.

Make sure you don't miss out on this year's big event.

You can either register online at www.ects.com or fax the coupon opposite for complimentary registration forms.

Check out the ECTS 97 website www.ects.com for news updates, press releases, full exhibitor listings, visitor and CTW premier club registration.

M A B C D E F G H I J K L M N O P Q R S

Fax back this coupon to **+44 (0) 1203 426 456**

☐ Please send me _____ visitor registration forms

☐ Please send me details, including fees, on the CTW Premier Club

Mr/Ms _____ Forename _____

Surname _____

Job Title _____

Organisation _____

Address _____

Postcode _____

Tel: _____

Fax: _____

Or post this coupon to:
ECTS 97, CDS, Curriers Close, Tile Hill, Coventry
CV4 8AW, UK.

sponsored by

CTW



un Miller Freeman
A United News & Media company

CREATING A GREAT DESIGN DOCUMENT

58

I've got to get product out. In the panic and dizziness, my head smashes against the CRT and next thing I know this genie whiffs up out of a virtual bronze-texture-mapped lamp and offers me three wishes. Without missing a beat, I answer, "I need..."

☞ A great team of talented, skilled, and dedicated engineers and artists (including a very understanding wife) with strong interpersonal skills.

☞ Enough time and money to allow for a mess-up or two.

☞ A first class design document.

Once upon a time, when coding a game involved one programmer (and *maybe* an artist) with a take-it-as-you-go budget and a loose deadline, documentation didn't need to be taken so seriously. You knew

B Y T Z V I F R E E M A N

what you wanted to make and you made it. If there were a few major changes along the way, the only one to complain was you. Nowadays, a thorough and readable document can mean the difference between a swift descent to budgetless Hell and a smooth ride to shrinked-wrapped Nirvana.

How the System Works

Most games go through three development stages, from concept to design to production. Think of them as “flash,” “paper,” and “grind.”

In the first stage, the concept paper acts both as a letter to yourself — setting out your goals clearly so you won’t lose sight of them — and as a sales tool for whomever takes the product to market down the road. Sometimes, this stage involves a working mini-prototype as well, which gives you a chance to experiment and revise your ideas.

The intermediate stage of design involves a lot of discussions with artists, animators, musicians, and engineers — trying things out, and finding ways to organize and set down your ideas.

In the final stage, production management is often left up to some expert in moving trains and tracks without major collisions. The original designer may be an integral part of the team, but in many cases — especially in large companies — the designer ends up as a kind of outside consultant.

Without question, the design document is where the original parent of the project exercises the most influence on how this little baby is going to grow up. Even if you, the designer, have decided to double as project manager, don’t delude yourself into thinking that you hold all the reins. A complex project involves many talented people. Skilled programmers and artists tend to have

minds of their own. While you intend to create a horse, the artist may be envisioning a unicorn and the programmer a highly efficient camel. A good document ensures that you are all planning to make the same thing. A *great* document ensures you all have the same feel for the inner soul of this thing. Think of it as a big band jazz score — it puts everybody’s mind in the same place, even when there’s still plenty of room for the stars to improvise.

Your document is a sort of intermediary between your mind and the real world. It ensures that what you have in mind is something that the real world is able to handle, and that what you end up with will be what you originally had in mind.

Finally, remember the adage to which any salty gamer will attest: “Great art is in the details.” Brilliant details flow naturally from the general gestalt as though they were present in that first flash of inspiration. But once you get into the hands-on implementation, it’s easy to lose that spark.

The Challenge

Prototyping parts of the project yourself is definitely a good idea — make whatever rough sketches you can. But again, it’s those details that count. The more details your imagination can hold, the greater a masterpiece your work will be.

Working from a document has a flip side, as well. Developing an exciting game has to be exciting. Some of the best parts of many projects were discovered in the heat of last-minute deadline panic. True, the pressures of

time and cost budgeting don’t allow for perpetual reiteration of concept, but you simply cannot expect a killer game to come out of dry, predictable work. The challenge is to create a design document that will allow your project to tolerate surprise adaptations without losing the integrity of its original direction and scope.

Ten Points for a Successful Design Document

1. DESCRIBE NOT JUST THE BODY, BUT THE SOUL.

If game development was just an automated input/output issue — something like writing code and being able to predict how it’s going to work — you could get by with a dry, descriptive document. The reality is that development is done by people, many of them creative people, who have their own minds; most will want to leave a stamp of that mind on everything they do.

It works like this: You provide specs to the artists and discuss with them what to do. You then visit the programmers and go over their specs. Both groups nod to everything you say.

That night, around 2AM, just as the constellation of C++ is rising in the west, the programmer reaches a mid-life crisis and begins to think, “What, a geek programmer the rest of my life? Is this what my mother expected from me? Why, I can design a game just as well as anybody else!” And the hands keep typing code.

Around the same time, the artist has just woken up before his machine, having fallen into a deep stupor while waiting for a complex 3D rendering to finish. Unsure and not really caring

TABLE 1. The three stages of documentation.

Documentation Stage	Contents	Purpose
Concept Paper	Genre; target audience; description; most compelling features; market information; cost and time to develop.	It defines the concept, scope, worthiness and feasibility; sells the idea to your client, publisher, employer, and venture capitalist.
Design Document	Description of the body and soul of the entire project, with all the details, and the method by which each element will be implemented.	It ensures that what is produced is what you want to produce.
Production Documents	Time-management charts (Gantt, PERT, and so on); task database; budget spreadsheet; technical specifications; Q/A database.	It implements the design document on time and within budget.

Experience the be-all, end-all online resource for
the game development community...
gamasutra.com



Gamasutra

The Art and Science of Making Games

COMPUTER
GAME
DEVELOPERS
CONFERENCE™

ON THE FRONT LINE OF GAME INNOVATION
Game
DEVELOPER

COMPUTER
GAME
DEVELOPERS
ASSOCIATION™

whether he's dreaming or is actually getting paid for all this, immersed in that wild world of artistic genius where fantasy and reality blend as one, the phosphors come together in ways previously unimagined — certainly not by you.

By the next morning, your horse has become a unicorn with two humps. With creative people, instructing is not good enough. You need to inspire.

In your design document, don't satisfy yourself with a detailed description of every article and nuance. Take time to describe the feel that the game should have, the purpose behind each element, the experience each user will have, and any other aspects of the game's soul you can envision and describe.

For example, say you're designing a shooter. You want to train your players to deal with certain challenges before they actually meet them, so you place less lethal mini-challenges a few steps in advance. You're going to have to explain that to everybody on the development team, so they'll understand why certain things are where they are and why they work the way they do. That way, even if (read: when) your team toys with and mangles your ideas as they exist on paper, you can still harbor hopes that the outcome will have the same or similar overall effect. Or maybe even better.

2. MAKE IT READABLE. Go ahead, provide your people with full pages of 10-point, sans serif, 80-characters-per-line text, and demand that they read it. You may want to bundle Advil in the package — for those who actually take the pains to obey orders.

I try to follow at least some of the guidelines of good page layout.

- Plenty of white space
- Serif font for body text
- Bold headers
- Spaces between paragraphs
- Short lines of text
- Direct the eye towards important material
- Use a hierarchical, "2D" format (see what I wrote about outliners in the "Design Documentation Tools" sidebar)

Many instances call for a table, spreadsheet, or chart. Use them and make them sensibly attractive.

3. PRIORITIZE. Now that you realize that you're working with other conscious egos, you'll appreciate the urgency of

tagging certain game elements as sacred. True, there are no guarantees, but if you use the tag sparsely enough, it may get some respect. But don't stop there. As long as you're tagging ideas, you'll also want to distinguish between things that you intend to do and things that you'd *like* to do if time, budget, skill sets, and technicalities permit.

Even when you provide animations and prototypes, put the concept into words as well. True, an animation is worth a gigabyte of words, but words can communicate in ways that animations can't.

Then there's the trash bin — things that sounded great, but were trashed for good reason. Refer to them explicitly and explain the reason that they were trashed. If you don't, I can almost guarantee that they will be resurrected. Here's your list of tags:

- Indispensable
- Important
- If Possible
- Rejected

You may wish to use visual symbols to represent these. Don't rely on color, since documents aren't always printed in color.

4. GET INTO THE DETAILS. A document without details is useless. Generalities can be interpreted by anybody in any way that they like. "Thou Shalt Not Kill" meant one thing to Moses and another to a Spanish conquistador. Detailing whom you shouldn't kill and under which circumstances would have been more helpful. The same holds true for your document: Once you've described some practical details and given some examples, your idea becomes more concrete — and harder to shove around.

For example, don't just say, "Bronze bird is invincible." Describe exactly what happens to this creature in each possible instance of its being hit, and how it recovers afterward. True, if the animator has any spunk and artistic dignity, you can rest assured that he won't follow your specifications. But at least he'll have a clear idea of what you want to achieve, and his modifications won't seriously alter the related portions of the game.

Don't just say, "At this point, users will have to press the jump key with the arrow key to climb the wall." Describe what will happen if a player tries anything else. Explain why you think users will be able to figure out the combination that you've provided. Explain what about the environment suggests that it's possible to climb this wall.

Again, your artist will come up with something else, perhaps something even more suitable than what you originally conceived. That's real success: When your developers' results come out even closer to your original flash of conception than what you were able to describe on paper. But this won't happen unless you first lucidly describe your concept.

Don't just say, "Bongo Man is stronger than Bongo Boy, but Boy has faster reflexes." Use tables, spreadsheets, and charts to assign real values to the character's speed of movement, how many hits the character can take, how much damage the character's hits do, how many cels it takes to animate a hit, and so on. This sort of spreadsheet is invaluable in the Q/A and tweeking stages of production.

Don't just say, "Most people will figure out the whole game in a few days." Make a chart of predicted product life in different households, indicating at which points in time you expect various features to be discovered. User testing will later provide valuable feedback for designing your next game.

5. SOME THINGS MUST BE DEMONSTRATED. Sometimes a few rough sketches are enough, but if the idea is truly important to your concept of the project, you may want to make a rough animation yourself. When behaviors of elements become too complex and ambiguous to describe on paper, you'll want to make a prototype. A side benefit of prototyping is that this practice often leads to a simpler, more elegant solution.

You couldn't be in 25 places at once, but your **Computer Game Developers' Conference CD-ROM** was.

The complete CGDC Conference and Expo captured in all its crazed glory—the next best thing to being there.

THE CONFERENCE

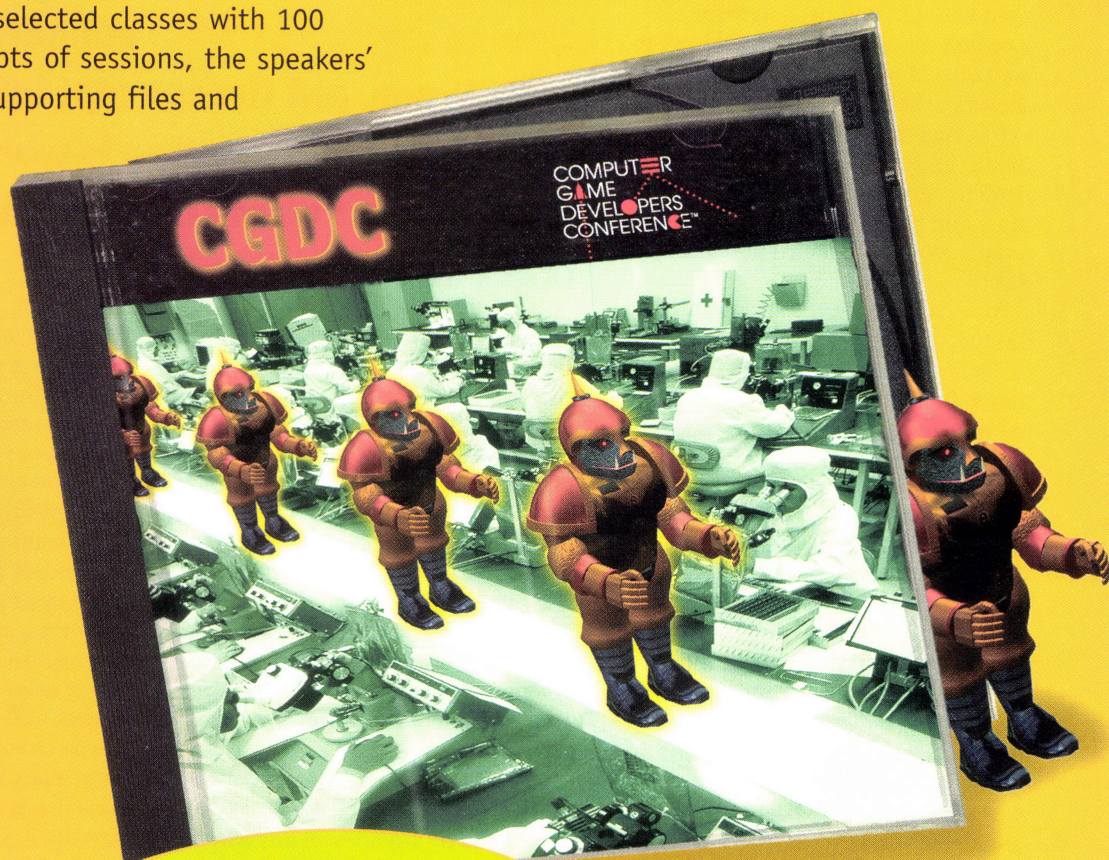
Don't worry if you had to cut class—the CGDC CD-ROM was taking notes for you. The disc includes video and audio clips of selected classes with 100 papers, over 100 transcripts of sessions, the speakers' presentation materials, supporting files and software demos.

THE EXPO

Too many booths, too little time... catch the listings of over 150 companies offering the latest and greatest development tools and technologies. Visit your favorite booth again and again. Toss out your classifieds—check out job listings of all the major job companies looking to hire.

THE MADNESS

Event highlights let you feel like you're still in the middle of it all: Stealth scenes from the schmooze fest. The calm before the hospitality suite storm. The Carnival documented. Views from the rescue helicopters. A glimpse of the ugly aftermath.



MISS THE CONFERENCE?
Feeling left out? Attend the virtual conference and expo—order your very own CD-ROM today.

www.cgdc.com

THE DEBATE

Virtual roundtables carry on all the discussions started at the CGDC—don't miss the chance to have the last word. Links to the CGDC website let you add your \$0.02 worth to this ongoing discussion with all your new and old friends from the CGDC.

**ALL THIS FOR
ONLY \$199...**

Call 800.444.4881 or 913.841.1631
Fax 913.841.2624 • Email orders@mfi.com

Even when you provide animations and prototypes, put the concept in words as well. True, an animation is worth a gigabyte of words, but words can communicate in ways that animations can't. Words also clearly spell out the vital nuances that may be missed when watching the animation.

6. NOT JUST "WHAT" BUT "HOW." In the real world, the "how" determines the "what." For example, suppose you've opted for claymation. Work out the process of how the images will be captured and *document everything*. What material and what color should the backdrop be? What camera should be used and why? What are the steps for processing the captured frames? And on and on. If you've tried it, you'll know that any one of these factors can have a serious impact on the end result.

Or suppose you're working on a motorcycle racing game. You state that the motorbikes must be balanced by their differing pros and cons. You even provide a chart that shows how balanced they are. Then you state that tweaking will be necessary. State *how* you plan to tweak — what is the process? Suppose the main character in your game is the Phantom of the Opera. Describe how the player's keyboard is mapped as a pipe organ. Provide a map of each key. Specify how many channels of sound will be available. Talk it over with your programmer and work out every detail of *how*. Then document it. Two different "hows" can mean two very different results.

7. PROVIDE ALTERNATIVES. Project managers spend a lot of time with their Gantt charts and PERTs. Personally, I can't really say that this stuff is effective for game development — principally because there are just so many unknowables. The more radical and pioneering your game technology, the less predictable the development stream is going to be. The best thing you can do to ensure that your team reaches your milestones on schedule is to provide more than one way of doing things.

Lets go back to the keyboard as pipe organ example. Your engineer describes to you the ultimate method of getting awesome and funky results with tremendous power and depth to the user — at a cost of about 50 person-hours to implement. As with everything else we've discussed, you docu-

Design Documentation Tools

1. Lots of paper and pencils (nothing digital has yet replaced paper and pencil for hacking out ideas).
2. Even more erasers.
3. A good integrated document application that includes features such as:
 - a. A good outliner. Most decent word processors have an outline mode. It's amazing how few people ever try out this feature. Some word processors will even automatically number and sub-number your items for you on the fly. An outliner makes it much easier to create documents in a nonlinear fashion, jumping around and inserting and moving information from place to place. An outliner also makes it easy to produce what I call "two-dimensional documents" — meaning documents that can be read both vertically and horizontally. Moving the eye vertically, a reader gets a feel for the contents and major concerns. Moving horizontally provides the reader with detailed information.
 - b. The ability to link illustrations to text.
 - c. Cross-reference updating.
 - d. Spreadsheets.
 - e. Tables.
 - f. Hot links are great.

I recommend Nisus Writer (Paragon Concepts) as the most powerful document processing tool, but ClarisWorks (Claris Corp.) is still the best integrated application available for both Mac and Windows. There are all sorts of project management tools available. I reserve those for the next stage of development.

4. A prototyping tool. mTROPOLIS, from mFactory, is a killer for determining behaviors of objects and getting things to happen really quickly. Macromedia Director is still the standard for quick and dirty animations. You may also need a 3D modeler, such as 3D Studio.
5. A basic library of sound effects. You don't have to make the final decision, but you'll probably want to try a few things yourself and give your audio engineer some idea of what you want.
6. Some pieces of art representative of the style that you intend to use. You'll probably need to make a list and use it to work with an artist while you are composing your document.
7. A knowledgeable friend to read the thing over thoroughly once it's finished and point out the remissions, contradictions, and ambiguities that every author fails to notice.

ment the whole thing.

You can't stop there. You've got to ask, "What would it take if we just wanted a trimmed-down, eight-channel pipe-organ? And what will we need to achieve the bare minimum? And what if we just had some assistant doing this?" And then you document all that as well. When the FedEx truck is on its way over for the final daily pickup, you'll be able to save your skin with a simple, "OK, do Plan C."

One of the biggest mistakes I've made in product design is asking engineers, "Can it be done?" Unless you're asking a first-class programmer, the question is useless. More specifically, responses fall into one of three categories:

(Lousy programmer) "Sure, that's no

problem."

(Mediocre programmer) "Nope. Can't be done."

(First-class programmer) "I could do it like this and it'll take two weeks. Or I could make a slight modification like this and it'll take five hours."

Always ask for more than one alternative and an estimate of how long each will take. Then indicate your preference — do this if we have time, or this if we don't.

8. GIVE IT A LIFE. I've already warned you against strangling the inspiration and spontaneous creativity that comes with the excitement and fun of seeing ideas become living objects in your hands. You've got to allow your document to tolerate change — by you or by (hopefully intelligent) others.

I first learned this lesson as a music composition major at the University of British Columbia. With much toil, I had written a neo-renaissance brass quintet of which I was quite proud. My professor liked it, too. When we brought it to the college's star brass quintet for rehearsal, however, I passed through several stages of horror, disbe-

lief, indignation, and clinical depression within ten seconds. The quintet began to play, then stopped on signal from the tuba player. The fellow took out his pencil and began to change a few notes, and then everyone continued. It happened more than once.

My professor, noting my sudden faint state of health, turned to me and

commented, "Don't worry, they did that to Mozart as well. And they're usually right."

The fact is, no matter how good something looks on paper, the greatest expert still modifies things when it enters the concrete world of objective perception. Nevertheless, you don't want to witness the ruthless rape of your design and dreams. Rather, you're hoping for a kind of organic growth — ideas growing naturally out of the seeds you've planted without needing foreign limbs and bodies grafted onto them.

Here are some tips for creating a document that can tolerate change without corrupting the original idea or sabotaging the development process:

- Make certain to engrave in stone those aspects that are so essential to the game concept that they must not be changed.
- Make certain everybody understands the feel that the game is supposed to have and the purpose of each of its details.
- If information is repeated, it must be cross-referenced. Otherwise, if there are changes, you can end up with contradictory instructions.

And here are some tips for the actual implementation stage:

- When a change is suggested, check back in your design document and see if it is in concordance with the "soul" of your game.
- Check whether this is just an isolated change, or it's of major global ecological impact (see "The Ecology of Improvement"). If it's the latter, save it for your next project.
- Update the design document and include the reasons for the change. Or if you didn't make the change, say so and explain why it was rejected.
- Changes, deletions, and rejected ideas should be retained in a master document to avoid discussing the same thing twice.
- Everyone must be working from the same version. Past versions should be destroyed.
- *Crucial, Vital, and Urgent:* The design document must be maintained under one person's supervision only.

9. NOBODY SHOULD BE ABLE TO SAY, "I DID IT THAT WAY BECAUSE I COULDN'T FIND ANY REFERENCE TO IT IN THE DOCUMENT." I've seen documents that didn't even have

The Ecology of Improvement a.k.a. The Freeman Elegance-Gestalt Principle

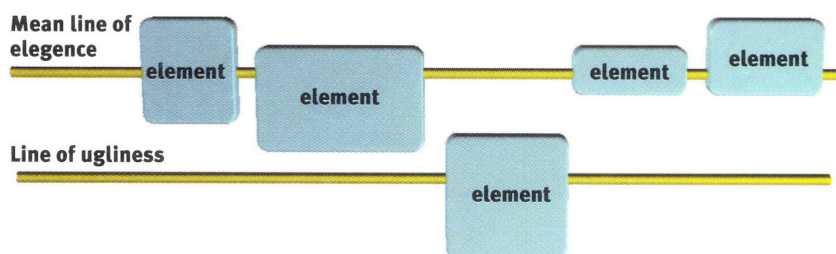
Improving the elegance of one part of an entity without addressing the gestalt of the whole will negatively effect the perceived elegance of every other part of that entity.

For example, you just had the leaks in your car's brake system patched, causing such pressure that your whole worn-out brake system blows out on you. Or you just bought a new pair of jogging shoes, which make your previously sufficient blue jeans look plain tacky.

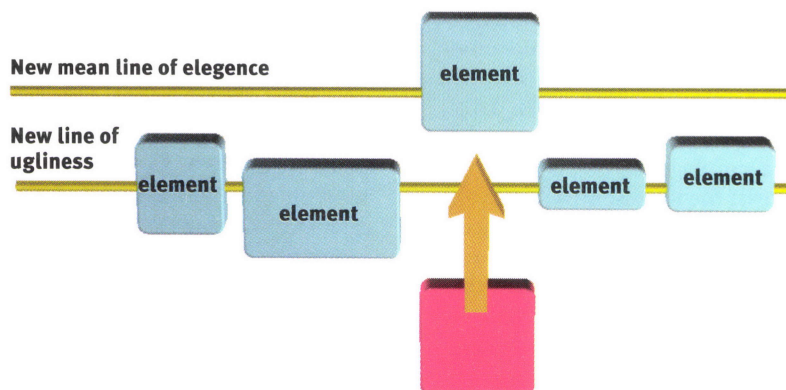
Or you just found a way to add realism to the movement of one of your characters, rendering all other characters jerky and sick in comparison. "Amazing," you mutter, that pale look of what-are-we-going-to-do-now-really-quickly all over your face. "They all looked great yesterday."

The lesson: Don't consider change in one area without first considering its ramifications on every other part of your design.

- 1 One element of your game is conspicuously inferior.



- 2 Somebody figures out a way to wildly improve that one element. Now many elements of your game appear conspicuously inferior.



Anatomy of a Design Document

Here's a suggested structure for your document. In actuality, you'll need to tailor the structure to suit your particular project. But my guess is that this example is pretty close to the industry standard. If you're working with a team that you have a history with, or if the game is following an established precedent, your document may be as short as 30 or so pages. That's short. Often, these documents can end up in the hundreds of pages. They're getting longer and longer as the industry matures.

Overview (sort of a recap and revision of the original concept paper)

The User Experience

- To what genre does this game belong? (We haven't really defined genres yet in this industry, as they have in Hollywood, so you'll have to be a little more specific. Best to describe similar products.)
- What part(s) of the brain are you attacking? (Reflex response? Imagination? Problem solving? Strategy?)
- What are the most compelling aspects of this game? (Give this section much consideration. It is the core of your "mission statement.")
- How deep is the product? (Is this a one-shot deal for buyers, or something they can keep getting into for a while?)

The Platform

- Arcade, home, or school?
- PC, Mac, console, or multiplatform?

The Users

- General audience
- Base target audience
- Describe typical users

Time

- Game-play time (This is one of the most crucial, decisive issues in the design of any game. It's impossible to make meaningful design decisions without establishing predicted maximum, minimum, and mean game-play times first.)
- Product life (How many days/months/years do you expect the user to keep coming back to your game? This period is usually based on how long it will take users to figure everything out and master it.)

Basic Concepts (gives a feel for the game, why things are the way they are, and what the essential, indispensable elements are)

Storyline

- The background story (if applicable)
- Storyline or object of the game play
- Rules of the game (Justify these rules and explain what you expect to see from them.)

Heroes and Villains

- Include biographical information and descriptions, even if this won't be implemented in the software. This will be of great help to the artists and animators.

Novelties and Compelling Features

- This is your chance to state the things you could not bear to see disappear from this project, and justify your emotional attachment to them.

Navigation Chart (an illustration of how parts of the game link to each other)

Entry and exit

Main menu

Level movement

Access to preferences and credits

Global Behaviors (ensures that your game will have a consistent feel to it, avoids serious run-ins with the programmers)

Run through all the standard elements of your project (sprites, buttons, life-bars, input devices, and so on) and describe their behaviors in every circumstance you can imagine. Your programmers will shower you with wreaths for this one. Later, you can go change things on them — as long as the objects remain consistent, and everything is justified.

Illustrate the motion of each animation, at least in stick form. For 2D scrollers, fighters, and the like, you'll want to describe things cel by cel. With other projects, rough sketches of general movements and their approximate duration in microseconds may be enough.

If you are relying on a specific input device, justify your button-mapping and button-combination decisions.

Scenes and Action (In an adventure game, this will take up most of your document)

Include preferences, credits, and main menu.

In subchapters, lay out consistent behaviors of local elements.

Very often, a storyboard (that is, a series of panels illustrating each scenario) is provided. In many projects, however, this is clumsy and impractical.

Lists of Resources

You'll have to go over this with a fine-tooth comb to make sure it's thorough. Leaving out even a few items, or failing to describe them clearly, could prove a major source of exasperation later on.

This section comprises detailed lists of animations, sounds, music, narration, sprites, backgrounds — everything that needs to be created besides code.

Cross reference each item to its main reference in the document.

the pages numbered. And then they complain that people didn't follow instructions. Every good word processor will auto-number pages and print the date and title in the header or footer of every page. Some will even allow

out HTML. But remember that more often than not, people prefer to work from a hard copy. (That way there's something to read while rebooting after the hourly system crash.)

10. DELIVER IT IN GOOD CONDITION. After all

there are updates, provide *everyone* with the revised pages. At some points, you may need to provide new books *and throw out the old ones*.

You may wish to write your document using HTML and provide hot links, but remember that more often than not, people prefer to work from a hard copy.

you to change the header at new chapters. Use bold text to direct attention to important material. Repeat yourself in different parts of the document as much as you like, as long as you cross-reference so you can update everything together as well. Make a thorough Table of Contents.

You may wish to write your document using HTML and provide hot links. Some progressive word processors provide hot link capabilities with-

this, you need to do whatever you can to facilitate everyone actually reading and using the thing. A pile of papers doesn't get read — it doesn't look important enough. Only things with hard covers look important.

Create a list of everyone who is supposed to have a copy. Keep the list. Print out the whole thing with the date in the header of each page. Have holes made and put it in binders. Label the spine and cover of each binder. When

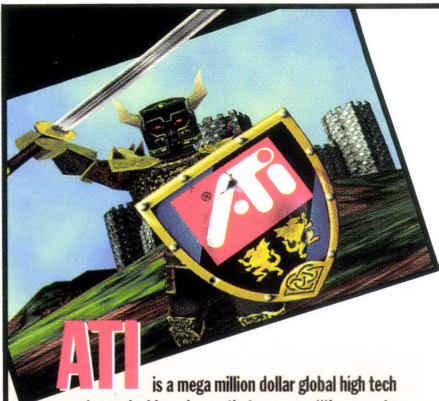
In Sum Check...

Movie makers use move scripts. Architects use blueprints. Musicians use a score. According to ancient hearsay evidence, even the Cosmic Creator created a design document — which He later let a few prophets take a peek at — before the primal "Let there be light!" So game developers, following their Supernal Role Model, can certainly do the same. Do it right and it's smooth sailing the rest of the way. ■

Tzvi Freeman teaches Game Design and Documentation at Digipen School of Computer Gaming in Vancouver, British Columbia, Canada. He has designed several commercial games and has acted as a consultant on many others. He can be reached at TzviF@aol.com.

ADVERTISER INDEX

NAME	PAGE	NAME	PAGE
3D Labs	20	Multigen	29
3Name 3D	41	NuVision Technology	39
AlphaPowered	C2-1	Numerical Design	17
ATI Research Inc.	67	Quantum3D	30
Caligari Corp.	45	Rad Game Tools Inc.	C4
CGDC on CD ROM	62	Red Storm Entertainment	67
Conitec Datensysteme	41	Scitech Software	35
Diamond Multimedia	9	Silicon Graphics	15
Diamondware	48	Softimage/Microsoft	7
Electric Image Inc.	2	Sonic Foundry	47
Immersion Corp.	25	Techweb	23
Immersion Corp.	49	Tritech	4-5
InstallShield Corp.	C3	Web '97	53
MPG '97	55		



warning: The following ad contains graphic material

HARDWARE

*Hardware Engineering Manager
System Architect - PC, WS, Multimedia Exp.
Senior Hardware Engineer - Scan & DFT
Hardware Engineer - Design Verification
Hardware Engineers - System
Senior Hardware Engineer - I/O
Senior Hardware Engineer - Audio/Modem*

SOFTWARE

*Software System Architect - Win 95, Multimedia
Senior Software Engineer - Graphics Drivers
Senior Software Engineer - Performance
Software Engineer - Performance
Senior Software Engineer - Internet Applications
Senior Software Engineer - Game Development*

MARKETING

Product Marketing Engineers/Managers

QA

*QA Release Engineer
QA Technician*

DEVELOPER SUPPORT & GAME PORTERS

*Developer Relations Professionals
Senior Software Engineers for Game Porting
Software Engineer For Windows95 Demo Creation
Junior Demo Engineers
OpenGL Software Engineer For Demos
Open GL Demo & QA Technician*

ATI is a mega million dollar global high tech player. Insiders know that we are "the one to beat" when it comes to 3D graphics and multimedia accelerator cards and chips. Without ATI, computer games and internet sites would be flat and a lot less fun. When the enemy gets "fried" or when an evil empire goes up in flames, when avatars dance and environments dazzle - we make it happen.

We've developed relationships with industry superstars like Microsoft, IBM, Sony, Gateway, NEC, Fujitsu, Acer, Apple, and Toshiba, and together, we have developed advanced business graphics and game platforms that exploit ATI's 3D graphics technology. The result is gaming with more reality, more detail, more interactivity, more challenge, more fulfillment . . . you get the picture.

Sound interesting? Like the idea of working on "out there" projects with the masters of the game? If so, bring your genius and skills our way and you will work on products that ship in the millions of units — and that add excitement to millions of lives. We have the following openings in our Marlboro, MA division:

If you're ready to dive into a whole new 3D world, at our Marlboro, MA division, send, fax, or E-mail your resume to:

**ATI Research, Inc.,
Human Resources, Dept. GDM,
4 Mount Royal Avenue,
Marlboro, MA 01752-1978,
Fax (508) 303-3920,
E-mail: jobs@atitech.com.**

No agencies or phone calls please.

<http://www.atitech.ca>
Check us out!



Join The Masters of the 3D Game.

Looking to Hire Professional Game Developers?

Take advantage of *Game Developer's* powerful position in the game developer market place. Your company can be on the frontlines of the game developer community with a response-oriented advertisement in the "Splash Screen" section of *Game Developer* magazine. Your 1/8 page, 4-color advertisement will reach over **25,000 Game Developer** subscribers who look to the "Splash Screens" section for jobs, products, and services. Don't miss out on this incredible opportunity to hire your technical professional developers. **Please contact Chris Cooper at 415-908-6614 or by e-mail at ccooper@mfi.com with all your recruiting needs.**

Red Storm ENTERTAINMENT

**Ready to work on projects where you make a difference?
Ready to build a company that rewards hard work with excellent benefits and stock options? If so, take this opportunity to join Tom Clancy and the rest of Red Storm.**

We're looking for enthusiastic, game savvy software engineers, artists, QA analysts, and producers who want to make great games.

**Please send your cover letter and resume to:
Human Resources, Red Storm Entertainment,
114 MacKenan Drive, Suite, 100, Cary, NC 27511
FAX: 919 460-4530.**

**Job descriptions available at
www.redstorm.com**

Accept the Challenge!



Escaping the Solo Trap

I'm aware of only one "multiplayer-only" game that was published by a major publisher — my own **ROBOT RASCALS** by Electronic Arts. There's a reason. Solo sells. Or at least it has until now. We

are witnessing the birth of a new games medium — online games — that will be about people playing together. To be part of this new medium, we game designers are going to have to overcome the old system built around sequels and solo play. Most of us will be glad to be doing original work, but I believe we will first have to realize how obsolete many of our previous design instincts have become in this new online world.

All of our features, imagery, and concepts as designers have gone to support the single-player game. But solo games have a wholly different style than the people-oriented games that will make up the online games medium. Solo games concentrated on entertaining just one user, so all the resources of the machine were devoted to that end.

Brilliant graphics and sounds were used to set the scene for the player — making his (the market was 90% male in the solo world, but online it's more like 60%) experience more compelling. Segues and cut scenes were triggered as appropriate to that audience of one. No provision needed to be made as to what the other players would be doing while the awards ceremony visuals were running. In solo games, the "other players" were all AI stand-ins who would mindlessly wait while the player gorged on eye candy. And the pandering didn't stop with the player's ego either; solo games needed to push the platform as well, since one of the biggest perks for many hardcore players was showing off their new hardware.

The features of a solo game were geared to take advantage of the "learning curve" involved in the process of mastering a new game and its environment. Normally, there were whole groups of features (new levels, stronger monsters, and so on) that only showed up in certain environments as the play-



er advanced. This type of feature-heavy design also had an advantage over solo games besides player titillation. Designers could hide the limitations of the AI opponent behind a veil of added game elements that kept challenging the player when the opponent couldn't.

Hiding the limitations of the artificial opponent led to a whole group of limitations in games. Although subtle nuances of pattern recognition might be trivial for a human player, even the most obvious pattern presents a nightmare for an AI. The code involved in teaching a war game AI to identify the "front" is incredibly complex anyway — imagine if some of the units are "unseen" or of unknown strength. Thus, the tendency was to make the externals of solo games very conventional (if not simplistic). Designers also discarded opportunities for interactions with subtle audio-visual cues in favor of algorithmic and concrete presentations. Rather than allude to something with patterns or sounds (the identification of which humans excel at), designers used the same sort of logic in their visual and audio representations as they used in their artificial opponent's analysis of the

world. There were no "maybes." There was only "true" and "false," zero and one. In addition, the need for competent AI required that the internal models be computational (which computers "love") rather than heuristic (which humans enjoy). In **M.U.L.E.**, I varied the output of land if the adjacent plot was producing the same thing. This feature gave the AI fits, but it presented an interesting trade-off in land acquisition for human players. The better answer for a solo design would have been to use some simple mathematical equation.

Taken together, these design elements may make good solo games, but they conspire to make games that are poorly suited to humans playing with humans. Rather than "over the top" production values, online games will reward small downloads, good multiplayer balance, and smooth play experiences. These emphases will preclude pauses for "cut-scenes" and pandering to a single player's ego or hardware vanity. Since a person derives so much challenge from anticipating another's actions, complex feature sets aren't needed. Simple sets of rules that are consistent over time make multiplayer games more accessible. However, subtle nuances in the audio/visual presentation of products make for much richer play experiences for human opponents. And, finally, heuristic (rules-of-thumb-based) models are much more appealing to players than complex numeric systems, playing to the strengths of our brains without sacrificing playability. To be part of this new medium, designers will have to abandon the solo model for design. If we can do this, we will be a long way toward making the kind of products that the online world will reward. ■

Dani Bunten Berry designed and directed the development of twelve original computer games. Among them were ten multiplayer games and three online games, including the first modem-to-modem game (MODEM WARS) and the first four-player network game (GLOBAL CONQUEST). Her best known titles were COMMAND HQ, SEVEN CITIES OF GOLD, and M.U.L.E. She is currently a design consultant.

Just how **extreme** is *new* InstallShield5 Professional?

InstallShield5 Professional is an installation development system that combines an easy-to-use visual development environment with the reliability and horsepower of proven InstallShield technology. It's the only installation development system that uses visual tools to give complete and total flexibility to the creation of your application's installation. Two years in the making, new InstallShield5 Professional was designed for those developers with need for a proven, world-class installation toolkit.

NEW FEATURES INCLUDE:

- Totally integrated 32-bit visual development environment
- Powerful, fully integrated and automated media builder
- Drag-and-drop visual file layout allows for easy modification and updates
- Full power of InstallScript for customized installation
- Color syntax editing
- Visual editors for File properties, Groups, Components and Setup Types
- Support for integrating .AVI video, WAVE/MIDI sound, 256-color images
- Support for global distribution

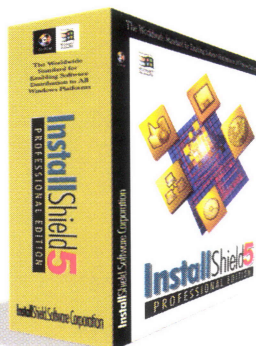
800-496-1955

www.installshield.com
info@installshield.com

InstallShield® Software Corporation
900 National Parkway, Suite 125, Schaumburg, IL 60173 USA

InstallShield® Software Corporation

© Copyright 1997 InstallShield Software Corporation. All rights reserved. InstallShield is a registered trademark of InstallShield Corporation. Windows is a registered trademark of Microsoft Corporation. 0397



The Best in Game Development Technology!

Smacker Video Technology



Smacker is a compressor for video, animation and sound data designed specifically for games. Smacker has been used in all aspects of game design: cinematics, cut-scenes, video-sprites, transparent videos, image decompression, scrolling video backgrounds, and more. It has been used in over 400 games because it is fast (damn fast!), easy-to-implement, and available for most game platforms.

The Smacker SDK is available for DOS, 16-bit Windows, Windows 95, Windows NT, Win32, Mac, and PowerMac. The Smacker SDK API is identical across the platforms and includes everything necessary to playback videos with synchronized sound.

For graphics, Smacker has built-in support for VESA 1.x (direct decompression into banked video RAM) and VESA 2.0 (linear frame buffer support). Under Windows, Smacker supports WinG, CreateDIBSection, DispDIB, and DirectDraw. For the Mac, Smacker supports both GWorlds (with lightning-fast assembly blitters to augment CopyBits) and direct-to-screen decompression. Since Smacker can decompress into any linear piece of 8-bit or 16-bit (new!) memory, using it with your own graphics code or a third party library (such as MGL) is no problem.

For sound, Smacker has built-in support for the Miles Sound System for DOS and Windows, DirectSound for Windows 95 and NT, the Windows waveOut system, HMI's SOS library for DOS, Diamondware's STK, and Sound Manager for the Macintosh.

Smacker supports Watcom C for DOS, any development system that supports DLLs for Windows (Visual C++, Borland C, Watcom C, Borland Delphi, etc.), and CodeWarrior for Macintosh. There is also a full-featured Smacker Xtra for Macromedia Director.

Partial list of Smacker and/or Miles Sound System customers:

Accolade, Activision, Acclaim, Boffo, Microsoft, Electronic Arts, Bullfrog, Lucas Arts, MindScape, Maxis, Core Design, Dynamix, Zombie, Viacom, Hypnotix, Papyrus, Blizzard Entertainment, Impressions, Take 2 Interactive, Blue Byte, Animatek, Stormfront Studios, GT Interactive, Hasbro, Krisalis, Sierra, Sir-Tech, Byron Preiss, New World Computing, Orient Vision, Sony, Attic Entertainment, Illumina, Atari, Time Warner, Software Sorcery, 47 Tek, N Space, McGraw Hill, Looking Glass, Capstone, Spectrum Holobyte, IntraCorp, Strategic Simulations, Software 2000, Compton's New Media, Rainbow America, Entertainer, Virgin Interactive, Criterion, Interactive Magic, SCI Limited, Trilobyte, Bethesda, Firestorm, Adeline, Psygnosis, Rocker Science, Epic, Legend, Merit, MECC, Creative Insights, Velocity, Origin, Sculptured Software, Software Toolworks, Capcom, and Sega.

... over 1,200 games - see our web-site for title lists!

The Miles Sound System



Cool Digital Sound Features:

Multiple channel mixing (limited only by CPU power), volume control, panning, pitch shifting, hard drive or CD-ROM streaming playback, on-the-fly format conversion, and much, much more!

Awesome MIDI Features:

Looping, branching, triggered digital sound playback, tempo control, powerful MIDI callbacks, etc.

Other Features:

Extensive timer library, complete Red Book CD-audio support, built-in Win32s support, full DirectSound support, complete Smacker support - play multiple sound effects and video sound tracks simultaneously!

Supported Platforms:

DOS with Watcom or Borland. Windows, Win32s, and Windows 95 with any development environment that supports DLLs.

The Smacker utilities are now FREE!!
Simply download them from our web-site
at www.radgametools.com!

The Smacker SDK and the Miles Sound System
are licensed on a per-product or per-site
basis with NO ROYALTIES WHATSOEVER!

Call for discounts when you license both
Smacker & the Miles Sound System together.

801-322-4300

850 South Main Street
Salt Lake City, UT 84101

801-322-4300

FAX 801-359-6169

CIS: 73237,75

www.radgametools.com



G A M E T O O L S